Prosessitietomallin toteutus graafitietokannassa

Nina Tyni

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Ohjaaja: Marko Junkkari
Toukokuu 2015

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Nina Tyni: Prosessitietomallin toteutus graafitietokannassa
Pro gradu -tutkielma, 58 sivua
Toukokuu 2015

Tässä tutkielmassa käsitellään jäljitettävyyttä ja toimitusketjujen mallintamista graafitietokannassa. Esimerkkinä toimitusketjujen mallintamisesta käytetään jäljitettävyysgraafia. Tutkielmassa esitellään jäljitettävyysgraafin toteutus Neo4j-graafitietokannassa. Graafitietokanta on yhdistetty Lucene-käänteishakemistoon, mikä mahdollistaa jäljitettävyystietojen yhdistämisen dokumentteihin kohdistuvaan sanahakuun. Tavoitteena on tutkia, miten jäljitettävyysgraafin pohjalta luodusta tietokannasta ja siihen yhdistetystä käänteishakemistosta voidaan hakea prosesseihin ja tuotteisiin liittyvää tietoa. Tätä varten on ohjelmoitu Java-ohjelmointikielellä hakuohjelma, jonka kautta tehtävillä esimerkkikyselyillä tiedonhakua havainnollistetaan.

Avainsanat: tietomalli, graafitietokanta, jäljitettävyys, jäljitettävyysgraafi.

Sisällys

1. Johdanto

Nykyään yleisesti käytössä olevat tietokantajärjestelmät on suunniteltu perinteiseen liiketoimintaan liittyvän tiedon käsittelyyn, esimerkiksi pankkitoimintaa ja varastonhallintaa varten. Tiedon käsittely- ja tallennusvaatimukset ovat kuitenkin muuttuneet huomattavasti ajoista, jolloin nämä järjestelmät on kehitetty. Muun muassa Internetin julkisen käytön lisääntymisen ja verkkotekniikan kehittymisen myötä tallennetun tiedon määrä on kasvanut valtavasti etenkin 2000-luvulla, ja tiedon rakenne on muuttunut. Monilla sovellusalueilla, esimerkiksi sosiaalisessa mediassa, biologiassa ja paikannuksessa, käsitellään monimutkaisia olioita, joilla on keskenään monimutkaisia suhteita, eivätkä perinteiset tietomallit enää riitä tällaisen tiedon käsittelyyn. [Grolinger *et al.*, 2013]

Perinteisten tietomallien rinnalle on kehitetty uusia tietomalleja, jotka pyrkivät ratkaisemaan ongelmia, joita tiedon määrä ja rakenne aiheuttavat vanhoissa järjestelmissä. Yksi viime vuosina kiinnostuksen kohteina olleista malleista on graafitietomalli. Graafitietomalli itsessään on jo 1980-luvulla kehitetty tietomalli, mutta siihen perustuvia järjestelmiä on alettu kehittää laajemmin vasta viime vuosina [Robinson *et al.*, 2013]. Kiinnostus graafitietomallia kohtaan on palannut, koska monissa sovelluksissa käsiteltävä tieto on luonnostaan verkkomaista, eli se sisältää runsaasti tietoa kuvaavien olioiden keskinäisiä suhteita. Esimerkiksi sosiaalisissa verkostoissa, verkkokauppojen suositusjärjestelmissä ja paikannusjärjestelmissä olioiden keskinäiset suhteet ovat vähintään yhtä tärkeitä kuin itse oliot tai niiden tietoalkioiden arvot.

Tässä työssä esimerkkinä verkkomaisesta tiedosta käytetään jäljitettävyysjärjestelmiä, joilla mallinnetaan tuotteiden toimitusketjuja. Toimitusketjujen mallintamisen tavoitteena on kerätä tietoa tuotteiden valmistuksesta: niiden valmistukseen käytetyistä materiaaleista ja niiden reitistä toimitusketjussa. Toimitusketjuista muodostuu helposti laajoja verkostoja, koska tuotetta ja sen komponentteja käsitellään usein monissa eri prosesseissa, joista vastaavat eri toimijat [Blackhurst *et al.*, 2005].

Jäljitettävyysjärjestelmien kehittämisestä ja toimitusketjujen hallinnasta on tullut tärkeä osa yritysten laadunhallintaa [Moe, 1998]. Esimerkiksi ongelmatilanteissa on tärkeää, että tuotteen alkuperä voidaan jäljittää luotettavasti. Jäljitettävyys voi olla myös tärkeä markkinointiväline, koska varsinkin kuluttajien vaatimukset tuotteiden alkuperän ilmoittamisesta ovat lisääntyneet voimakkaasti. Kuluttajia kiinnostavat tuotteeseen ja tuotantoketjuun liittyvät eettiset tekijät, kuten toimittajien taustat, materiaalien alkuperä ja tuotteen ympäristövaikutukset, mutta toisaalta myös tuotteen turvallisuus. Esimerkiksi elintarvikkeiden laatu ja turvallisuus herättivät runsaasti keskustelua alkuvuodesta 2013, kun useaan Euroopan maahan levinneessä niin kutsutussa hevosenlihaskandaalissa havaittiin, että tietyn lihantoimittajan tuotteista löytyi hevosenlihaa, jota ei tuoteselosteessa mainittu [Yle, 2013]. Autovalmistajat puolestaan ovat joutuneet kutsumaan jopa miljoonia autoja korjattavaksi, kun autoissa on havaittu vikoja, jotka voivat vaikuttaa autojen turvallisuuteen [Helsingin Sanomat, 2014; Taloussanomat, 2013].

Tässä työssä esitellään tuotteiden toimitusketjujen mallintamiseen tarkoitettu jäljitettävyysgraafi. Jäljitettävyysgraafi on prosessimalli, joka on tarkoitettu fyysisten tuotteiden

valmistusprosessien ja prosesseihin liittyvien raaka-aine- ja informaatiovirtojen mallintamiseen [Junkkari and Sirkka, 2011a]. Sen avulla voidaan myös kohdentaa prosessiin liittyviä ominaisuuksia, kuten prosessissa aiheutuneita kustannuksia, prosessissa käsitellyille tuotteille.

Jäljitettävyysgraafi esitellään T-paidan valmistusta kuvaavan esimerkkitapauksen avulla. Esimerkkitapauksen pohjalta on luotu graafitietomalliin pohjautuva tietokanta, jonka avulla kuvataan, miten jäljitettävyysgraafi voidaan esittää graafitietokannassa. Tietokannan rinnalle on ohjelmoitu hakuohjelma, jolla voidaan hakea tietoa tietokantaan tallennetuista prosesseista ja tuotteista. Hakuohjelmassa on graafinen käyttöliittymä, jonka kautta tietokantakyselyt tehdään ja jossa tulokset esitetään. Graafinen käyttöliittymä mahdollistaa myös hakutuloksiin liittyvien tietojen visualisoinnin.

Jäljitettävyysgraafissa prosessien välillä siirtyviin informaatiovirtoihin voi kuulua myös erilaisia dokumentteja. Siksi hakuohjelmaan on toteutettu graafitietokannan rinnalle käänteishakemisto, jonka avulla tiedonhaku voidaan ulottaa tietokannan lisäksi dokumentteihin. Hakuohjelmassa graafitietokanta ja käänteishakemisto on yhdistetty niin, että sanahaku kohdistetaan samalla sekä tietokantaan että käänteishakemistoon.

Työn toisessa luvussa esitellään aluksi, miten tietoa kuvaillaan erilaisilla tietomalleilla. Tarkoituksena on esitellä erilaisia tiedon kuvaamiseen käytettäviä käsitteitä ja rakenteita. Kolmannessa luvussa esitellään graafitietomalli ja siihen perustuva Neo4j-graafitietokanta, jota tässä työssä on käytetty. Neljännessä luvussa käsitellään ensin yleisesti fyysisten tuotteiden jäljitettävyyttä, toimitusketjuja ja jäljitettävyysjärjestelmiä. Sen jälkeen esitellään toimitusketjujen mallintamiseen käytettävä jäljitettävyysgraafi. Neljännessä luvussa esitellään myös tässä työssä käytettävä T-paidan valmistusta kuvaava esimerkkitapaus. Viidennessä luvussa kuvataan jäljitettävyysgraafin toteutus graafitietokannassa. Kuudennessa luvussa kerrotaan, millaisia tietoja järjestelmällä voidaan hakea. Seitsemännessä luvussa on kuvaus graafitietokannan ja sen oheen ohjelmoidun hakusovelluksen teknisestä toteutuksesta. Tutkielman yhteenveto on kahdeksannessa luvussa, ja lopuksi yhdeksännessä luvussa esitellään johtopäätökset.

2. Tietomallit

Tietokantaa ja sen sisältämiä tietoja voidaan kuvata käyttämällä kolmea abstraktiotasoa: fyysistä tasoa, käsitetasoa ja ulkoista tasoa [Elmasri and Navathe, 1994]. Fyysinen taso tarkoittaa tapaa, jolla tietokanta on tallennettu fyysiselle tallennusvälineelle. Se käsittää esimerkiksi tiedostot ja niiden käsittelyssä käytettävät indeksit ja muut tallennusrakenteet. Käsitteellinen taso kuvaa koko tietokannan rakenteen fyysistä tasoa abstraktimmalla tasolla. Käsitteellisellä tasolla kuvataan, millaisia tietoja tietokannassa käsitellään: millaisia käsiteltävät kohteet ovat, miten ne liittyvät toisiinsa ja millaisia rajoituksia niihin liittyy. Ulkoiseen tasoon liittyvät näkymät, joilla voidaan esittää valittuja osia tietokannan sisällöstä.

Tiedon kuvaamiseen käytetään eri abstraktiotasoilla eri tietomalleja. Tietomalli tarkoittaa tiedon kuvaamiseen käytettävää käsitteistöä, jolla määritellään tiedon rakenne ja usein myös tiedon käsittelyssä käytettävät operaatiot [Elmasri and Navathe, 1994]. Eri tietomalleissa käytetään erilaisia primitiivejä rakenteen määrittelyyn, ja eri tietomallit eroavatkin toisistaan siinä, millaisia yksityiskohtia niillä voidaan ilmaista ja miten tiedot ja erityisesti niiden väliset suhteet esitetään [Silberschatz *et al.*, 1996]. Esimerkiksi käsitteellisellä tasolla käytetään käsitteellisiä malleja, jotka kuvaavat tiedon hyvin yleisellä tasolla. Fyysisen tason kuvaamiseen käytettävät fyysiset tietomallit puolestaan määrittelevät tietyn tietokannan sisäiset tieto- ja tiedostorakenteet. Näiden väliin jäävät loogiset tietomallit ovat määrättyyn tietokannanhallintajärjestelmään sopivia malleja, joiden perusteella tietokanta voidaan muodostaa [Elmasri and Navathe, 1994].

Käsitteelliset mallit kuvaavat kohdealueen huomioimatta varsinaisen tietokantatoteutuksen yksityiskohtia. Kuvauksessa käytettävät käsitteet ovat helposti useimpien käyttäjien ymmärrettävissä: tosimaailman kohteet esitetään olioina, niiden ominaisuuksina ja olioiden välisinä suhteina. Käsitteelliseen tasoon liittyy käsitteellinen kaavio, jossa kuvataan kohdealueen oliotyypit, olioiden väliset suhteet ja niihin liittyvät rajoitteet. Käsitteellinen kaavio kuvataan usein käyttämällä tietokannanhallintajärjestelmästä riippumatonta korkean tason tietomallia eli käsitteellistä tietomallia [Elmasri and Navathe, 1994]. Tällaisia käsitteellisiä malleja ovat esimerkiksi ER-malli [Chen, 1976], semanttiset tietomallit, kuten SDM [Hammer and McLeod, 1981], ja funktionaaliset tietomallit [Sibley and Kerschberg, 1977].

Käsitteellisen tietomallin avulla laadittu kaavio voidaan muuntaa tietokantakaavioksi, joka kuvataan tietokannanhallintajärjestelmässä käytettävällä loogisella tietomallilla ja tiedonmäärittelykielellä. Loogisia tietomalleja ovat esimerkiksi verkkomalli [DBTG, 1971], hierarkkinen malli [Tsichritzis and Lochovsky, 1976] ja relaatiomalli [Codd, 1970]. Myös oliomalli lasketaan loogiseksi tietomalliksi, joskin Elmasri ja Navathe [1994] huomauttavat, että se on lähellä abstraktimpia käsitteellisiä tietomalleja ja sitä käytetään usein käsitteelliseen mallintamiseen etenkin ohjelmistotuotannossa.

Abstraktiotason lisäksi tietomallit voidaan luokitella tietueperusteisiin (record-based) ja olioperusteisiin (object-based) tietomalleihin sen mukaan, millaisina rakenteina ne esittävät tiedon. Relaatiomallia, hierarkkista mallia ja verkkomallia kutsutaan tietueperusteisiksi tietomalleiksi,

koska ne kokoavat tiedon tietueisiin. Olioperusteisissa malleissa, kuten ER- ja oliomallissa, tieto puolestaan kootaan olioihin, joilla on ominaisuuksia ja suhteita. [Elmasri and Navathe, 1994] Ullman [1988] tosin jaottelee mallit arvo- ja olioperusteisiin sen perusteella, tukevatko ne olioidentiteettiä. Ullman katsoo myös hierarkkisen ja verkkomallin olevan olioperusteisia, koska niihin perustuvien järjestelmien tietueilla voidaan ajatella olevan fyysisestä tiedostorakenteesta johtuva identiteetti, joka erottaa kaksi samat arvot omaavaa tietuetta toisistaan.

Tässä työssä käsitteellisistä tietomalleista esitellään ER-malli (kohta 2.1), koska se on käytetyin käsitteellinen tietomalli ja siinä käytettäviä peruskäsitteitä voi käyttää kuvaamaan myös monia muissa tietomalleissa esiintyviä käsitteitä. Loogisista tietomalleista esitellään relaatiomalli, verkkomalli, hierarkkinen malli ja oliomalli (alakohdat 2.2–2.5). Hierarkkinen ja verkkomalli ovat esimerkkejä varhaisista tietomalleista, jotka perustuivat lähinnä tietorakenteiden määrittelyyn tiedostojärjestelmissä. Fyysisen ja loogisen tason erottamiseen perustuva relaatiomalli puolestaan on mukana, koska se suosituin looginen tietomalli. Oliomalli esitellään, koska sen tapa kuvata tosimaailman oliot tietokantaolioina muistuttaa graafitietomallin tapaa kuvata oliot ja niiden suhteet hyvin samanlaisina verkostoina kuin mitä ne muodostavat tosimaailmassa.

2.1. ER-malli

Chenin [1976] kehittämä ER-malli (entity-relationship model) on yleinen tietokantojen suunnittelussa käytettävä käsitteellinen tietomalli. ER-mallin peruskäsitteitä ovat olio (entity), suhde (relationship) ja attribuutti (attribute). Oliot ovat objekteja, jotka voidaan tunnistaa yksikäsitteisesti. Olio voi olla fyysinen kohde, kuten henkilö tai auto, tai käsitteellinen kohde, kuten yritys tai yliopiston kurssi.

Attribuutit ovat olioita kuvailevia ominaisuuksia. Oliot kuuluvat samaan oliotyyppiin, jos niiden attribuutit ovat samat. Yksittäisillä olioilla on kuitenkin jokaiselle attribuutille omat arvonsa, ja oliot erotetaan toisistaan attribuuttien arvojen perusteella. Attribuuttia tai attribuuttijoukkoa, jonka perusteella voidaan erottaa toisistaan kaikki oliojoukon oliot, kutsutaan avainattribuutiksi. [Elmasri and Navathe, 1994] Myös suhteilla voi olla attribuutteja [Chen, 1976]. Esimerkiksi yrityksen ja yrityksessä työskentelevän henkilön välisen suhteen attribuutti voisi olla päivämäärä, jolloin työsuhde alkoi.

Suhdetyyppi on oliotyyppien välinen riippuvuus tai muu yhteys, joka kertoo, miten oliotyypit liittyvät toisiinsa. Suhdetyypin ilmentymä, suhde, liittää toisiinsa olion kustakin suhteeseen osallistuvasta oliotyypistä. Kullakin suhdetyyppiin osallistuvalla oliotyypillä on suhteessa tietty rooli. Oliotyypin roolinimi kertoo, millainen rooli kyseisen oliotyypin olioilla on suhteessa. Jos suhdetyyppiin osallistuvat oliotyypit ovat eri oliotyyppejä (esimerkiksi ”yritys” ja ”työntekijä”), roolinimenä voidaan käyttää oliotyypin nimenä. Jos sama oliotyyppi sen sijaan esiintyy suhteessa useampaan kertaan, roolinimellä erotellaan kunkin samaa oliotyyppiä edustavan olion rooli suhteessa. Esimerkiksi kahden ”työntekijä”-tyyppisen olion suhteessa toisen rooli voi olla esimies ja toisen alainen. [Elmasri and Navathe, 1994]

Suhdetyyppeihin liittyy yleensä rajoituksia, joilla säännellään suhteeseen osallistuvia olioita. Yksi tällainen rajoitus on kardinaalisuus, joka ilmaisee, kuinka moneen suhteeseen olio voi osallistua samanaikaisesti ja kuinka monta oliota tiettyyn suhteeseen voi osallistua. Kardinaalisuus voi olla yhden suhde yhteen (1:1), yhden suhde moneen (1:n) tai monen suhde moneen (m:n). Yhden suhde yhteen -rajoitus tarkoittaa, että yksi olio voi liittyä vain yhteen olioon. Yhden suhde moneen -tyyppisessä suhteessa yksi olio voi liittyä moneen olioon. Monen suhde moneen -tyyppisessä suhteessa yhden oliotyypin olio voi liittyä moneen toisen oliotyypin olioon ja päinvastoin. [Elmasri and Navathe, 1994]

2.2. Relaatiomalli

Relaatiomalli [Codd, 1970] on looginen tietomalli, joka perustuu matemaattiseen relaation käsitteeseen. Se kuvaa tiedot ja niiden käsittelyn joukko-opin ja relaatioiden avulla niin, että tietojen rakenne voidaan esittää ottamatta kantaa varsinaiseen tekniseen toteutukseen. Suurin osa nykyään käytössä olevista tietokannanhallintajärjestelmistä perustuu relaatiomalliin.

Relaation havainnollistamiseen käytetään usein taulukkoesitystä. Siinä relaatio kuvataan taulukkona, jonka rivit kuvaavat tosimaailman olioon tai suhteeseen liittyviä arvoja. Taulukon ja sarakkeiden nimet kertovat, mitä arvot tarkoittavat. Relaatiomallin käsitteillä puhuttaessa rivit ovat monikkoja (tuple), sarakkeiden nimet attribuutteja (attribute) ja taulukko itsessään on relaatio (relation). Attribuuttiin liittyy aina arvojoukko (domain). [Elmasri and Navathe, 2010]

Relaatiomallissa tietokanta koostuu relaatioista. Relaatio on joukko monikkoja. Monikko puolestaan on järjestetty lista, jossa kukin arvo on tiettyyn arvojoukkoon kuuluva arvo. Arvojoukko on kokoelma atomisia arvoja, ja se kuvaa, minkä tyyppisiä arvoja attribuutilla voi olla. Matemaattisesti ilmaistuna relaatio on arvojoukoista muodostetun karteesisen tulon osajoukko. Arvojoukkojen ei tarvitse olla erillisiä, vaan eri attribuuteilla voi olla samantyyppisiä arvoja.

Relaation rakenne kuvataan relaatiokaaviolla (relation schema), joka koostuu relaation nimestä ja joukosta attribuutteja. [Elmasri and Navathe, 2010] Attribuuttien nimet kertovat, miten relaation monikkojen arvot tulkitaan. Esimerkiksi kirjoja ja niiden kirjoittajia sisältävä tietokanta voitaisiin kuvata seuraavilla relaatiokaavioilla:

Henkilö(Henkilönumero, Nimi, Syntymäaika)

Kirja(ISBN, Nimi, Julkaisuvuosi)

Kirjoitti(Kirjoittajan_numero, ISBN)

Yllä olevien relaatiokaavioiden mukaisiin relaatioihin voisi puolestaan kuulua seuraavanlaisia monikkoja:

(1, Ramez Elmasri, 20-10-1950)

(978-0136086208, Fundamentals of Database Systems (6th Edition), 2010)

(1, 978-0136086208)

Koska relaatio on matemaattisesti joukko ja joukon alkiot ovat erillisiä, kaikki relaation monikot ovat keskenään erilaisia. Relaatiossa ei siis voi olla kahta monikkoa, joilla on samat arvot kaikille attribuuteille. Näin monikot pystytään erottamaan toisistaan sisältönsä perusteella. Yleensä

monikkojen erotteluun ei kuitenkaan tarvita kaikkia attribuutteja, vaan relaatiossa on useampi avaimeksi (key) kutsuttu attribuutti tai attribuuttiyhdistelmä, jonka perusteella monikot voidaan yksilöidä ja josta ei voida poistaa yhtään attribuuttia niin, että jäljelle jäävät attribuutit riittäisivät yksilöimään monikot. [Elmasri and Navathe, 2010] Näistä avaimista valitaan pääavain (primary key), jonka arvon perusteella relaation monikot tunnistetaan. Pääavain merkitään relaatiokaavioon alleviivaamalla siihen kuuluvat attribuutit.

Pääavainta käytetään monikkojen yksilöinnin lisäksi myös relaatioiden välisten suhteiden esittämiseen. Relaatiosta viitataan toiseen relaatioon lisäämällä attribuutiksi sen relaation pääavain, johon viitataan. Viittaukseen käytettäviä avaimia kutsutaan vierasavaimiksi (foreign key). Vierasavaimella voidaan liittää toisiinsa myös saman relaation eri monikoita. Esimerkiksi relaatiokaaviossa Työntekijä(Henkilönumero, Nimi, Esimies→Työntekijä) Esimies-attribuutin arvona on kussakin monikossa jokin toinen työntekijä.

2.3. Verkkomalli

Verkkomalli on abstraktio tietokantojen fyysisen tason toteutuksessa käytettävistä käsitteistä, ja se liittyy läheisemmin tietokannan fyysisen tason rakenteeseen kuin esimerkiksi relaatiomalli, vaikka ne molemmat ovatkin loogisia tietomalleja. Verkkomallissa tietoalkiot kootaan tietuekokoelmiin, ja alkioiden väliset suhteet esitetään linkkeinä, jotka vastaavat fyysisen tason osoittimia [Silberschatz *et al.*, 1996].

Verkkomallissa tieto tallennetaan tietueisiin, jotka koostuvat toisiinsa liittyvistä tietoalkion arvoista. Tietueet luokitellaan tietuetyyppeihin. Tietuetyyppi kuvaa samantyyppistä tietoa sisältävien tietueiden rakenteen. Tietuetyypin kuvaus vastaa siis relaatiomallin relaatiokaaviota ja tietuetyypin esiintymä relaatiokaavion esiintymää, eli taulua. Tietuetyypillä on nimi, ja siihen liittyy joukko tietoalkioita. Kullakin tietoalkiolla puolestaan on nimi ja tietotyyppi. [Elmasri and Navathe, 1994]

Kokoelmatyyppi kuvaa kahden tietuetyypin välisen 1:n-suhdetyypin. Kokoelmatyyppi koostuu kokoelmatyypin nimestä, omistajatietueesta ja jäsentietueesta. Tietokannassa kokoelmatyyppejä vastaavat esiintymät. Jokaisessa esiintymässä yksi omistajatietuetta edustava tietue liitetään joukkoon jäsentietueita. Kokoelmatyypin esiintymä koostuu siten yhdestä omistajatietueesta ja jäsentietueista, joita voi olla useita tai ei yhtään.

Koska kokoelmatyypin omistajatietue voi olla omistajana vain yhdessä esiintymässä, yhdellä kokoelmatyypillä ei voida esittää m:n-suhdetta kahden tietuetyypin välillä. Sen sijaan m:n-suhteet esitetään verkkomallissa käyttämällä kahta kokoelmatyyppiä ja täydentävää tietuetyyppiä [Elmasri and Navathe, 1994].

2.4. Hierarkkinen malli

Hierarkkinen malli muistuttaa verkkomallia, paitsi että hierarkkisessa mallissa suhteiden on muodostettava puurakenne, kun taas verkkomalli sallii mielivaltaiset graafirakenteet [Silberschatz *et al.*, 1996]. Hierarkkisen mallin peruskäsitteet ovat samat kuin verkkomallissa: tietue, tietoalkion

arvo ja tietuetyyppien väliset suhteet. Tietue koostuu joukosta tietoalkion arvoja, ja samantyyppiset tietueet kuuluvat samaan tietuetyyppiin [Elmasri and Navathe, 1994].

Tietuetyyppien väliset suhteet esitetään vanhempi–lapsi-suhdetyypillä, joka on kahden tietuetyypin välinen 1:n-suhdetyyppi [Elmasri and Navathe, 1994]. Vanhempi–lapsi-suhdetyyppi vastaa verkkomallin kokoelmatyyppiä. Se muodostuu yhdestä vanhemmasta ja lapsista, joita voi olla useita tai ei yhtään. Kukin tietuetyyppi esiintyy lapsena tasan yhdessä vanhempi–lapsi-suhdetyypissä, paitsi hierarkian juuri, jolla ei ole vanhempaa.

Vanhempi–lapsi-suhteen rajoituksista johtuen m:n-suhdetyyppien esittäminen hierarkkisessa mallissa vaatii joko lapsitietueiden monistamista tai virtuaalisten vanhempi–lapsi-suhteiden käyttöä [Elmasri and Navathe, 1994]. Virtuaalisia suhteita käytettäessä tietokantaan luodaan useita hierarkioita ja suhteet esitetään lisäämällä hierarkioiden välille osoittimia. Yleensä osoitin lisätään virtuaalisesta lapsesta virtuaaliseen vanhempaan, mutta sen lisäksi on mahdollista luoda osoitin virtuaalisesta vanhemmasta linkitettyyn listaan, joka sisältää kaikki kyseisen vanhemman virtuaaliset lapset.

Koska hierarkkisessa mallissa tietuetyyppien väliset suhteet ovat aina 1:n-tyyppisiä, suhteiden esittäminen on huomattavasti rajatumpaa kuin esimerkiksi verkkomallissa [Elmasri and Navathe, 1994] vaikka rajoituksia voidaankin kiertää edellä mainituilla tavoilla. Verkkomallissa tietuetyyppi voi kuulua jäsenenä kuinka moneen tahansa kokoelmatyyppiin, kun taas hierarkkisessa mallissa tietuetyypillä voi olla vain yksi varsinainen vanhempi ja yksi virtuaalivanhempi. Siten hierarkkinen malli on luonnollinen tapa mallintaa lähinnä tilanteita, joissa esiintyy pääasiassa yksisuuntaisia 1:n-suhdetyyppejä.

2.5. Oliomalli

Oliomallia alettiin kehittää 1980-luvulla, kun havaittiin, että yleisesti käyttöön omaksuttu relaatiomalli ei sovi sovelluksiin, joissa käsitellään monimutkaisia toisista olioista koostuvia olioita tai kompleksisia tietoalkioita, kuten kuvia tai videoita. Tällaisia sovellusalueita ovat esimerkiksi tietokoneavusteinen suunnittelu ja valmistus (CAD/CAM), maantieteelliset tietojärjestelmät, tiedonhaku ja erilaiset multimediasovellukset. [Elmasri and Navathe, 2010]

Oliomalliin perustuvissa tietokantajärjestelmissä tiedon kuvaamiseen käytetään alun perin oliopohjaisia ohjelmointikieliä varten kehitettyjä käsitteitä [Silberschatz *et al.*, 1996]. Malli perustuu tiedon ja tietoa operoivan koodin kokoamiseen tietokantaolioon: tosimaailman oliot kuvataan tietokantaolioina, joilla on yksikäsitteinen identiteetti ja ominaisuuksia, jotka määrittävät olion tilan ja käyttäytymisen [Kim, 1990]. Olion tilalla tarkoitetaan joukkoa attribuutteja, jotka kuvaavat olion ominaisuuksia, ja käyttäytymisellä metodeja, joilla olion tilaa voidaan muokata ja tiedustella.

Olion attribuuttien arvot tallennetaan olion sisäisiin ilmentymämuuttujiin (instance variable). Ilmentymämuuttujan arvo on yleensä itsekin olio tai joukko olioita [Kim, 1990], tai se voi olla yksittäinen arvo, arvojoukko tai kompleksinen arvotyyppi [Elmasri and Navathe, 2010]. Tämä

mahdollistaa monimutkaisten, toisista olioista koostuvien olioiden käsittelyn. Olioon tallennetuilla olioviittauksella voidaan myös esittää olioiden väliset suhteet.

Olion tilaan ja käyttäytymiseen liittyy kapseloinnin (encapsulation) käsite. Kapseloinnilla voidaan tarkoittaa sekä olioon liittyvien attribuuttien että metodien kokoamista samaan olioon [Kim, 1990] tai sitä, että olion ilmentymämuuttujat eivät ole suoraan käyttäjän käsiteltävissä [Elmasri and Navathe, 2010]. Kun ilmentymämuuttujat on suojattu piilottamalla ne käyttäjältä, niitä voidaan käsitellä olion ulkopuolelta vain ennalta määrättyjen metodien kautta. Metodeja voidaan käyttää lähettämällä oliolle kutsu, joka sisältää suoritettavan metodin nimen ja mahdolliset metodin tarvitsemat parametrit.

Oliomallissa tosimaailman olio ja sitä kuvaava tietokantaolio vastaavat suoraan toisiaan toisin kuin perinteisissä malleissa, joissa olioita koskeva tieto on usein hajautettu moniin relaatioihin tai tietueisiin [Elmasri and Navathe, 1994]. Oliopohjaisessa tietokantajärjestelmässä kullekin oliolle määrätään yksilöllinen tunniste, joka ei ole riippuvainen olion attribuuttien arvoista, joten olion identiteetti on pysyvä. Koska oliot tunnistetaan tämän erillisen tunnisteen avulla, olioilla, joilla on samat ominaisuudet, on silti erilliset identiteetit. [Elmasri and Navathe, 2010]

Oliot, joilla on keskenään samanlainen rakenne ja käyttäytyminen, kuuluvat samaan luokkaan. Luokka on siis eräänlainen objektin tyyppimäärittely samaan tapaan kuin verkkomallin tietuetyyppi tai ER-mallin oliotyyppi [Silberschatz *et al.*, 1996]. Luokan kuvauksella määritetään, millaisia muuttujia luokkaan kuuluvilla olioilla on ja miten näitä muuttujia voidaan käsitellä. Olio on luokan ilmentymä, ja olio luodaan aina yhteen luokkaan.

Olemassa olevista luokista voidaan johtaa uusia luokkia. Tätä kutsutaan periytymiseksi. Periyttämällä luotavaa uutta luokkaa kutsutaan aliluokaksi ja perittävää luokkaa yliluokaksi. Aliluokka perii kaikki yliluokkansa attribuutit ja metodit, minkä lisäksi aliluokalle voidaan luoda uusia ominaisuuksia. Luokalla voi olla kuinka monta tahansa aliluokkaa, kun taas yliluokkia voi järjestelmästä riippuen olla vain yksi tai useampia. Hierarkkisessa periytymisessä, jossa luokka perii ominaisuutensa vain yhdeltä luokalta, luokista muodostuu luokkahierarkia. Jos järjestelmä sallii moniperiytymisen, eli luokka voi periä ominaisuuksia useammalta yliluokalta, luokista muodostuu suunnattu graafi. [Kim, 1990]

3. Graafitietomalli ja Neo4j-tietokanta

Luvussa 2 esitellyistä tietomalleista verkkomalli ja hierarkkinen malli olivat aikanaan merkittäviä malleja, mutta nykyään käytössä enää hyvin harvassa, lähinnä päivittämättömässä järjestelmässä [Elmasri and Navathe, 2010]. Relaatiomalli sen sijaan on edelleen yksi käytetyimmistä malleista. Relaatiomalli on kuitenkin suunniteltu aikana, jolloin laitteistot ja ohjelmistot olivat hyvin erilaisia kuin nykyään, joten se ei sovellu kovin hyvin kaikista uusimmille sovellusalueille, joilla käsitellään lyhyessä ajassa valtavia määriä tietoja [Stonebraker *et al.*, 2007]. Siksi sen tilalle (tai rinnalle) on viime vuosina ehdotettu uusia tietomalleja, jotka on suunniteltu suurten ja rakenteeltaan monimutkaisten tietomassojen käsittelyyn. Yksi näistä malleista on graafitietomalli, joka tosin on vanha tietomalli, mutta jota on edellä mainituista syistä alettu tutkia uudelleen. Graafitietomalli muistuttaa verkkomallia, mutta linkkien sijaan yhteydet esitetään kaarina, jotka voivat olla kaksisuuntaisia. Tämä mahdollistaa verkkomallia joustavamman navigoinnin olioiden ja tietojen välillä.

Graafitietomalleja alettiin alun perin kehittää 1980-luvulla oliomallin rinnalla. Perinteiset tietomallit sopivat huonosti tilanteisiin, jossa tallennettava tieto on verkottunutta, eli se sisältää paljon graafirakenteita. Esimerkiksi maantieteellisissä tietojärjestelmissä tai hypertekstisovelluksissa tiedon keskinäiset linkitykset ovat tärkeitä, mutta arvo- ja olio-orientoituneet tietomallit on suunniteltu lähinnä tietoalkioiden arvojen tallentamisen näkökulmasta, ei suhteiden. Graafitietomalleja alettiin kehittää näiden suhteita koskevien rajoitusten poistamiseksi.

Yksi ensimmäisiä graafin ja tietokannan yhdistäviä ratkaisuja oli semanttinen verkko [Roussopoulos and Mylopoulos, 1975]. Semanttinen verkko on suunnattu graafi, jonka solmut edustavat tietokannan käsitteitä ja kaaret niiden semanttisia suhteita. Semanttisen verkon tarkoituksena oli täydentää relaatiomallia niin, että tietokannan tiedon merkitys voitaisiin sisällyttää itse tietokantaan. Graafia käytettiin siis tietokannan metatiedon tallentamiseen.

Itse tiedon tallentamista graafirakenteisiin ehdotti muun muassa Shipman [1981], jonka esittämän DAPLEX-kielen ja siihen liittyvän funktionaalisen tietomallin tarkoituksena oli mallintaa sovellusalue mahdollisimman samanlaisena kuin se esiintyy ihmisten ajatuksissa. Mallin peruskäsitteitä ovat oliot ja toiminnot, jotka liittävät oliot toisiinsa. LDM (Logical Data Model) -malli [Kuper and Vardi, 1984] puolestaan pyrki yleistämään relaatio-, hierarkkisen ja verkkomallin. LDM-mallissa tietokantakaavio koostuu suunnatusta graafista, jonka lehtisolmut (solmut, joista ei lähde yhtään kaarta) kuvaavat tietoa ja sisäsolmut kuvaavat tietojen välisiä yhteyksiä.

Graafitietomallit ja niiden kanssa samanaikaisesti kehitetyt oliomallit liittyvät myös toisiinsa, koska oliomallit hyödyntävät määrittelyissään graafirakenteita joko suoraan tai epäsuorasti [Angles and Gutierrez, 2008; Andries *et al.*, 1992]. Esimerkiksi O2-oliotietokannassa [Lécluse *et al.*, 1988] oliot muodostavat suunnatun graafin, ja GOOD (Graph-Oriented Object Data model) -mallissa [Gyssens *et al.*, 1990] tieto kootaan suunnattuun graafiin, jonka solmut tukevat olioidentiteettiä.

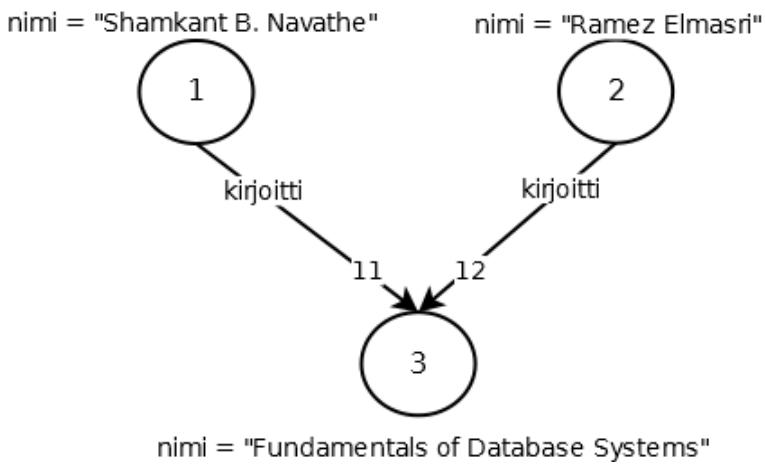
Graafitietomallien kehitys oli vilkkainta 1990-luvun puolessavälissä. Muut tietomallit, jotka oli kehitetty nimenomaan tiettyjen sovellusten tarpeisiin, erityisesti spatiaaliset, puolirakenteiset ja XML-tietomalli, kuitenkin syrjäyttivät silloisen kiinnostuksen graafitietomalleja kohtaan [Angles and Gutierrez, 2008].

Viime aikoina kiinnostus graafitietomalleja kohtaan on jälleen lisääntynyt osana kehitystä, jossa perinteisille tietokantajärjestelmille etsitään paremmin valtavien tietomäärien tallennukseen ja käsittelyyn sopivia vaihtoehtoja. Sen lisäksi, että tallennetun tiedon määrä on kasvanut 2000-luvulla valtavasti, tallennettavan tiedon rakenne on muuttunut [Batra and Tyagi, 2012]. Internet ja erityisesti sosiaaliset verkostot tuottavat paljon tietoa, joka sisältää runsaasti tietoa kuvaavien olioiden välisiä keskinäisiä yhteyksiä. Koska graafitietokannat on suunniteltu nimenomaan tällaisen verkkomaisen tiedon käsittelyyn, kiinnostus niiden kehittämiseen on palannut.

3.1. Graafitietomalli

Angles ja Gutierrez [2008] määrittelevät graafitietomallin mallina, jossa sekä tietokannan kaavio että esiintymät, tai vain jompikumpi, mallinnetaan suunnattuna graafina tai graafitietorakenteen yleistyksenä. Lisäksi tiedon manipulointi ilmaistaan graafeille ominaisilla operaatioilla ja graafirakenteelle voidaan asettaa tarvittaessa esimerkiksi eheyttä, tietokantakaavion ja -esiintymän vastaavuutta ja ominaisuuksia ja niiden arvoalueita koskevia rajoitteita. Graafitietomallissa oliot edustavat tosimaailman itsenäisiä yksiköitä. Suhde on ominaisuus tai predikaatti, joka osoittaa kahden tai useamman olion välillä olevan yhteyden.

Useimmat nykyisistä graafitietokannoista, kuten InfiniteGraph [Objectivity, 2015], Titan [Aurelius, 2015] ja Neo4j [Neo4j.com, 2015], perustuvat niin sanottuun ominaisuusgraafimalliin (property graph model) [Das *et al.*, 2014]. Kuvassa 1 on yksinkertainen esimerkki ominaisuusgraafista.



Kuva 1. Ominaisuusgraafi. Numerot 1, 2 ja 3 viittaavat solmujen tunnisteisiin ja numerot 11 ja 12 kaarien tunnisteisiin.

Ominaisuusgraafi koostuu joukosta solmuja ja kaaria. Jokaiseen solmuun liittyy yksilöllinen tunniste, joukko solmusta lähteviä kaaria, joukko solmuun saapuvia kaaria ja joukko avain–arvo-

tyyppisiä ominaisuuksia. Jokainen kaari yhdistää kaksi solmua toisiinsa. Kaaret ovat suunnattuja, eli niillä on lähtö- ja loppusolmu. Myös kaarilla on yksilölliset tunnisteet ja joukko avain–arvo-tyyppisiä ominaisuuksia. Lisäksi kaareen liittyy tunnus, joka kertoo, minkä tyyppinen suhde sen alku- ja loppusolmun välillä on.

Graafitietomalli sopii tilanteisiin, joissa pääpaino on käsitteiden ja niiden suhteiden kuvaamisessa. Kohdealueen tieto on tällöin verkkomaista, toisin sanoen se voidaan kuvata solmuina ja niiden välisinä yhteyksinä, ja on siten mallinnettavissa suoraan graafina. Esimerkiksi sosiaaliset verkostot, world wide web ja verkkokauppojen suosittelujärjestelmien pohjana olevat ostotiedot ovat sovellusalueita, joilla tieto on luontevaa mallintaa suhdeverkostoina[1]. Tällaisissa aineistossa tiedon sisäiset yhteydet ovat yhtä tärkeitä tai jopa tärkeämpiä kuin tietoalkioiden arvot: esimerkiksi tieto world wide webin sivustojen keskinäisistä linkityksistä on yhtä tärkeää kuin tieto siitä, mitä sivustoja ylipäänsä on olemassa.

Olioiden verkkomaisen järjestäytymisen lisäksi graafitietomallin käyttöä edellä luetelluilla aloilla puoltaa tapa, jolla tietoa analysoidaan. Aineistojen analysointia hallitsevat graafioperaatiot, kuten solmun naapureiden löytäminen, graafissa kulkeminen, lyhimpien polkujen löytäminen ja yhteisöjen havaitseminen [Dominguez-Sal *et al.*, 2010]. Esimerkiksi sosiaalisia verkostoja tutkittaessa kiinnostavaa voisi olla tieto kahden henkilön välisestä yhteydestä, jolloin graafimuodossa tallennetusta datasta voidaan etsiä, onko kyseisten henkilöiden välillä yhteys joko välittömänä suhteena tai onko henkilöiden välille löydettävissä polku muiden henkilöiden kautta.

Robinson ja muut [2013] mainitsevat graafitietomallia hyödyntävien tietokantojen eduksi erityisesti niiden suhteellisen vapaan rakenteen, jonka muokkaaminen on helppoa. Uudentyyppisiä suhteita, solmuja ja aligraafeja voidaan lisätä tarvittaessa ilman, että lisäykset häiritsevät vanhoja kyselyitä tai sovelluksen toimintaa. Kun tietokannan rakennetta voi muokata joustavasti, sovellusaluetta ei tarvitse mallintaa etukäteen niin tarkasti kuin perinteisissä tietomalleissa, mikä helpottaa tietokannan kehittämistä sitä mukaa kuin sovellukseen kohdistuvat vaatimukset tai tietämys kohdealueesta kehittyvät. Rakenteen muokattavuus tosin vaihtelee eri graafitietokantajärjestelmissä. Joissakin järjestelmissä tietokantakaaviota ei ole pakko määritellä ollenkaan, kun taas joissakin tietokantakaavion ja -esiintymän vastaavuudelle on asetettu tarkat rajoitteet [Angles and Gutierrez, 2008; Neo4j Manual, 2015b].

Vaikka graafitietomalli sopisi hyvin monelle sovellusalueelle, nykyään suosituimpia tietovarastoja ovat relaatiomalliin perustuvat tietokannat [Vicknair *et al.*, 2010]. Relaatiomalliin perustuvat tietokannanhallintajärjestelmät on kuitenkin suunniteltu tilanteisiin, joissa tietoalkiot ovat tyypillisesti pieniä ja joiden rakenne on yksinkertainen. Lisäksi tietoalkioiden käsittelyyn tarvittavat operaatiot ovat suhteellisen yksinkertaisia eivätkä yleensä vaadi rekursiivisia päättelyitä.

[1] Sosiaalisen verkostoitumisen sivustot, kuten Facebook, eivät tosin tällä hetkellä käytä graafitietomalliin perustuvia tietokantoja, vaan esimerkiksi relaatiotietokantoihin yhdistettyjä avain–arvo-pohjaisia tai sarakepohjaisia tietokantoja, koska ne sopivat ainakin tällä hetkellä paremmin massiivisten tieto- ja käyttäjämäärien käsittelyyn [Buerli, 2012]

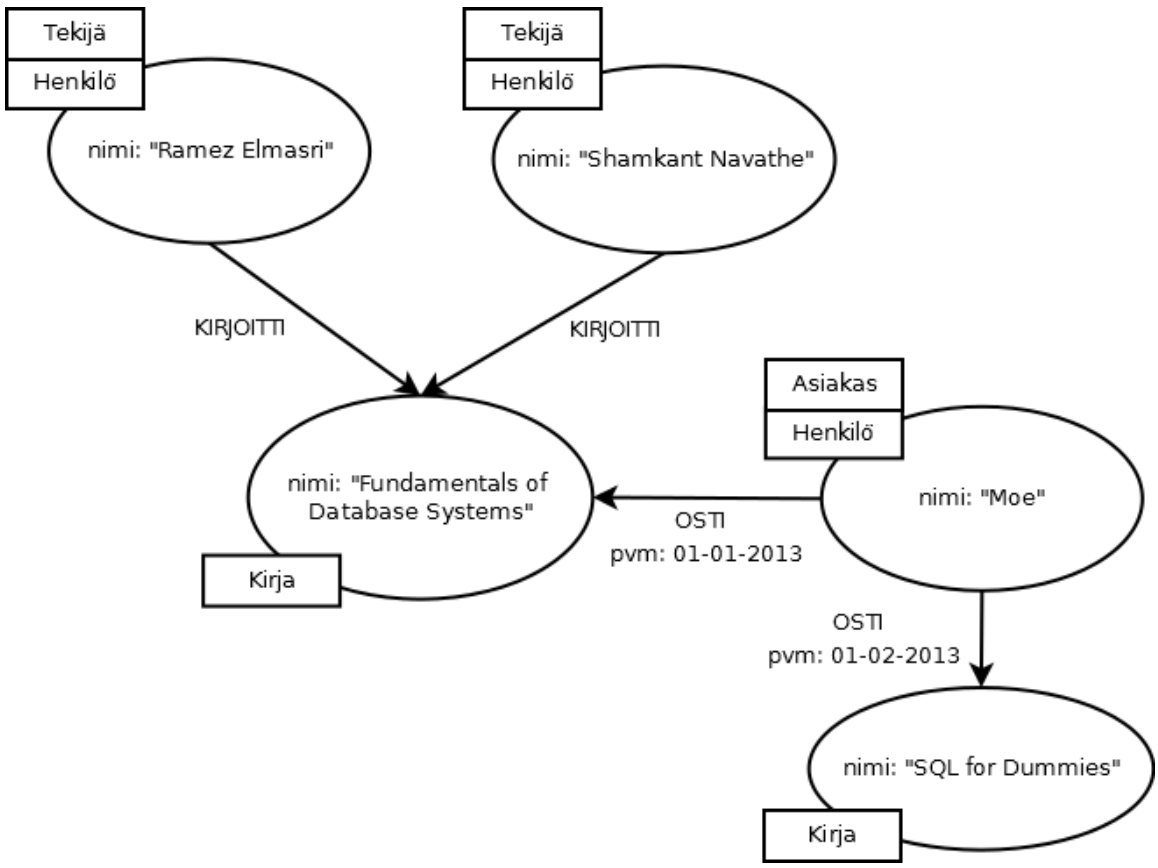
[Levene and Loizou, 1995] Jos sovellusalueen tieto kuitenkin on aiemman kuvauksen mukaisesti mallinnettavissa graafitietomallilla, olisi luontevaa valita tiedon tallentamista ja kyselyjä varten tietokanta, joka tukee suoraan graafeille ominaisia operaatioita, kuten graafissa kulkemista ja yhteisöjen etsimistä. Graafirakenteiden tallentaminen ja niiden käsittely onnistuvat relaatiotietokannassakin, mutta koska niitä ei ole suunniteltu graafien käsittelyyn, on käsittely usein tehotonta ja monimutkaista [Vicknair *et al.*, 2010].

3.2. Neo4j-graafitietokanta

Neo4j on Neo Technology -yhtiön kehittämä avoimen lähdekoodin tietokanta, jossa tieto tallennetaan edellisessä kohdassa kuvattuihin ominaisuusgraafeihin [Neo4j.com, 2015]. Se on toteutettu Java- ja Scala-ohjelmointikielillä, ja se pystyy skaalautumaan pienestä paikallisesta tietokannasta usealle palvelimelle hajautettuun järjestelmään. Neo4j-tietokanta julkaistiin alun perin vuonna 2007, ja se on tällä hetkellä suosituin graafitietokanta [DB-Engines.com, 2015].

3.2.1. Neo4j-tietokannan rakenne

Neo4j-tietokannassa graafi koostuu solmuista (nodes) ja suhteista (relationships), joilla on ominaisuuksia (properties). Tietokannan solmut vastaavat graafin solmuja ja suhteet kaaria. Ominaisuudet ovat avain–arvo-pareja, jotka voivat liittyä sekä solmuihin että suhteisiin. Kuvassa 2 on esimerkkigraafi, joka kuvaa yksinkertaista verkkokauppasovellusta.



Kuva 2. Esimerkkigraafi.

Solmut esitetään kuvassa soikioina ja suhteet niiden välisinä nuolina. Suorakulmiot ovat solmujen luokitteluun käytettäviä tunnuksia.

Solmut kuvaavat kohdealueen olioita ja asioita, joilla on ominaisuuksia ja suhteita muihin olioihin. Ominaisuudet ilmaistaan Neo4j-tietokannassa solmuihin liittyviä olioina tai arvoina. Solmut ovat suhteiden ohella Neo4j:n tietomallin tärkeimpiä tietoyksiköitä, ja solmuina mallinnetaan usein myös kohteita, jotka intuitiivisesti vaikuttavat suhteilta [Robinson *et al.*, 2013]. Esimerkiksi kuvan 2 OSTI-suhteen sijaan verkkokaupan ostotapahtumat voitaisiin esittää solmuina, jossa yksi verkkokauppatilaus on oma solmunsa, johon liitetään tilauksen tehneen henkilön lisäksi jokainen asia, joka kyseiseen tilaukseen kuuluu.

Suhde määritellään graafissa kahden solmun välille. Jokaisella suhteella on lähtö- ja loppusolmu, mistä seuraa, että tietokannan kaaret ovat aina suunnattuja [Neo4j API 2015a]. Suhteita ei kuitenkaan tarvitse määritellä molempiin suuntiin erikseen, koska graafissa kuljettaessa suhteet voidaan kulkea kumpaan suuntaan tahansa. Suhteen suunnalla onkin enemmän merkitystä tietokannan rakenteiden selventämisessä, ja suunta on aina mallintajan valinta. Kahden solmun välillä voi olla kuinka monta tahansa suhdetta, ja solmulla voi olla suhde myös itseensä.

Jokaisella suhteella on oltava tyyppi, joka kertoo, millainen suhde kahden solmun välillä on. Esimerkiksi verkkokauppaa kuvaavan graafin tapauksessa henkilösolmun ja kirjasolmun välinen OSTI-suhde tarkoittaa, että henkilö on ostanut kirjan. Suhdetyyppejä käytetään muun muassa graafissa kulkemisessa ja erilaisten solmujen tunnistamisessa. Esimerkiksi kaikki verkkokaupan asiakkaan ostamat asiat voitaisiin etsiä tietokannasta hakemalla solmut, jotka on liitetty asiakasta kuvaavaan solmuun OSTI-tyyppisellä suhteella. Suhdetyyppejä ei ole määritelty ennalta, vaan ne ovat vapaasti määriteltävissä tietokantaa rakennettaessa [Neo4j API 2015b]. Näin ollen sovelluksen suhdetyypit kertovat paljon graafin rakenteesta ja sisällöstä.

Ominaisuudet ovat avain–arvo-pareja, jotka voivat liittyä sekä solmuihin että suhteisiin. Avaimet ovat aina merkkijonomuodossa (Javan String-luokka), kun taas arvot voivat mitä tahansa Java-kielen perustietotyyppejä tai String-olioita tai taulukoita, jotka sisältävät perustietotyyppejä tai String-olioita. [Neo4j API 2015c] Ominaisuuksilla voidaan ilmaista solmujen ja suhteiden ominaisuuksia, kuten kirjan tai henkilön nimi tai kirjan ostoaika. Suhteiden ominaisuudet mahdollistavat myös esimerkiksi painotettujen graafien painoarvojen esittämisen.

Neo4j-tietokannalle voi määrittää kaavion, mutta se ei ole pakollinen [Neo4j Manual, 2015b]. Neo4j-tietokannassa kaavio rakentuu indekseistä ja graafille asetetuista rajoitteista. Näiden indeksien ja rajoitteiden määrittämiseen käytetään tunnuksia (label), joita voi määrittää solmuille. Tunnusten avulla solmuille voidaan määrittää rooli tai tyyppi ja niitä voidaan jakaa ryhmiin. Tunnuksia voi siis toisaalta ajatella esimerkiksi ER-mallin oliotyypin kaltaisena luokitteluna, mutta niitä voi käyttää muuhunkin kuin solmujen pysyvään luokitteluun. Koska tunnuksia voidaan lisätä ja poistaa ajonaikaisesti, niillä voi merkitä solmujen tilapäisiä tiloja. Yhdelle solmulle voidaan lisätä useita tunnuksia. Kuvan 2 esimerkkigraafissa tunnukset on merkitty solmujen päällä osittain olevilla suorakulmioilla.

3.2.2. Neo4j-tietokannan operaatiot

Tietokannan operaatioilla tarkoitetaan toimintoja, joilla tietokannan sisältöä muokataan ja haetaan. Tällaisia operaatioita ovat esimerkiksi tietojen lisääminen tietokantaan sekä niiden poistaminen, päivittäminen ja hakeminen. [Elmasri and Navathe, 1994] Neo4j mahdollistaa kaikki nämä operaatiot, mutta tässä niistä käsitellään vain tietojen hakemista.

Graafitietokannoille ominainen operaatio on kulkeminen (traversal). Graafissa kulkeminen tarkoittaa sen solmujen ja suhteiden järjestelmällistä läpikäyntiä annettujen sääntöjen mukaan. Neo4j-tietokannassa kulkemista varten määritellään lähtösolmu, minkä lisäksi voidaan määritellä esimerkiksi kulkujärjestys (syvyys- vai leveyshaku), minkä tyyppisiä ja suuntaisia suhteita seurataan ja kuinka pitkälle (syvyys) sekä mitkä solmut otetaan mukaan tulosjoukkoon [Neo4j Manual, 2015a]. Kulkemisen lopputuloksena on joukko solmuja, jotka täyttävät kulkemiselle määritellyt ehdot.

Esimerkiksi kuvan 2 graafissa kulkemista voitaisiin käyttää, kun halutaan etsiä muut kirjat, joita ”SQL for Dummies” -kirjan ostajat ovat ostaneet. Tällöin ”SQL for Dummies” -kirjaa kuvaava solmu olisi lähtösolmu, josta OSTI-tyyppisiä suhteita pitkin kulkemalla voidaan hakea kaikki kyseisen kirjan ostaneet henkilöt ja edelleen muut kirjat, joita nämä henkilöt ovat ostaneet.

Yleinen hakutarve on löytää tietokannasta solmu tai suhde jonkin ominaisuuden perusteella. Esimerkiksi graafissa kulkemiseen tarvitaan aina lähtösolmu, jonka on oltava tiedossa ennen kulkemista. Naiivi lähestymistapa hakuun olisi kulkea graafissa ja tutkia jokaisen graafin jäsenen kohdalla, toteutuuko hakuehto. Jos tietokannassa kuitenkin on paljon solmuja, on niiden läpikäyminen näin kulkemalla tehotonta. Koska tämäntyyppiset haut ovat tietokannassa hyvin yleisiä, niiden nopeuttamiseksi on Neo4j-tietokantaan toteutettu Apache Lucene -hakukirjastoon pohjautuva tietokannan indeksointi ja haku. Neo4j mahdollistaa sekä solmujen että suhteiden indeksoinnin.

Neo4j-tietokantaan liittyy myös Cypher-kyselykieli, jota voi käyttää graafiin kohdistuviin kyselyihin ja päivityksiin [Neo4j Manual, 2015c]. Se on deklaratiivinen kyselykieli, eli kyselyissä määritetään, mitä halutaan hakea, ei sitä miten hakutulos muodostetaan. Cypher-kyselyt muodostuvat lohkoista, joilla määritetään, mitä graafista haetaan, mitä ehtoja haettaville kohteille asetetaan ja mitä kysely palauttaa. Monet Cypherin avainsanoista on lainattu muista, vakiintuneista kyselykielistä. Esimerkiksi seuraava kysely palauttaa kuvan 2 esimerkkigraafista kaikki asiakkaat, jotka ovat ostaneet kirjan nimeltä SQL for Dummies.

```
MATCH (kirja {nimi:'SQL for Dummies'})<-[:OSTI]-(asiakas)
RETURN asiakas
```

Yllä olevan kyselyn MATCH-lohkossa määritellään rakenne, jota graafista etsitään. Solmut kuvataan kaarisulkeiden sisällä olevilla tunnisteilla. Niiden nimeämisellä ei ole merkitystä kyselyn suorittamisen kannalta, lähinnä kyselyn luettavuuden. Kaarisulkeiden sisällä olevissa aaltosulkeissa luetellaan ominaisuudet, jotka etsittävillä solmuilla halutaan olevan. Solmujen välissä oleva nuolimerkintä tarkoittaa, että graafista etsitään tietyn suuntaista ja tyyppistä suhdetta. Tässä

tapauksessa etsitään OSTI-tyyppistä suhdetta, jonka loppusolmulla on ominaisuus ”nimi”, jonka arvo on ”SQL for Dummies”. RETURN-lohkossa kerrotaan, että kyselyn halutaan palauttavan MATCH-lohkoa vastaavien rakenteiden alkusolmut.

4. Jäljitettävyys ja jäljitettävyysgraafi

Jäljitettävyys tarkoittaa tuotteiden kulun seuraamista tuottajilta kauppiaille ja kuluttajille. Sen on perinteisesti ajateltu olevan teollisuuden ja kaupan sisäisiin prosesseihin liittyvä käsite, joka liittyy tuotteiden alkuperän selvittämiseen ja esimerkiksi takaisinvetojen helpottamiseen ongelmatilanteissa. Nykyään jäljitettävyys kuitenkin kiinnostaa myös kuluttajia, ja siitä on tulossa yrityksille markkinointivaltti. Kuluttajat ovat yhä kiinnostuneempia ostamiensa tuotteiden alkuperästä ja turvallisuudesta, ja etenkin ruuan alkuperästä on tullut tärkeä ostoperuste [Saarinen, 2013]. Kuluttajia kiinnostavat esimerkiksi tuotteen raaka-aineiden alkuperä, tuotantoketjun eettisyys ja tuotteen ympäristövaikutukset.

Markkinoiden vaatimusten lisäksi jäljitettävyyttä edellyttää myös laki. Esimerkiksi Euroopan unionin tuoteturvallisuusdirektiivi [EUR-Lex, 2013a] edellyttää, että tuotetta tai tuotantoerää koskevat tiedot on ilmoitettava tuotteessa tai sen pakkauksessa. Myös valtioneuvoston kulutus-tavaroista ja kuluttajapalveluksista annettavia tietoja koskevassa asetuksessa [Tukes, 2013] säädetään, että tavarassa on tarvittaessa oltava valmistuserätunnus tai muu tieto, jonka avulla tavara voidaan tarvittaessa yksilöidä tai jäljittää. Euroopan parlamentin ja neuvoston vuonna 2002 antamassa elintarvikelainsäädäntöä koskevassa asetuksessa puolestaan määrätään, että kaikissa tuotanto-, jalostus- ja jakeluvaiheissa on huolehdittava siitä, että käytettävät raaka-aineet voidaan jäljittää toimitusketjussa yksi porras taaksepäin. [EUR-Lex, 2013b].

Jäljitettävyyden kehittäminen edellyttää, että tuotteiden tunnistukseen, prosessien ja ympäristöjen kuvailuun, tietojen tallennukseen, analysointiin ja siirtoon sekä järjestelmien yhdistämiseen on olemassa riittävät tekniset valmiudet. Tämä koskee sekä laitteistoja, kuten mittauslaitteita ja tunnistustekniikkaa, että ohjelmistoja, kuten tietokoneohjelmia ja tietojärjestelmiä. [Opara, 2003] Tarvittavien tietojärjestelmien kehittämiseksi on kuitenkin ensin pystyttävä luomaan riittävän tarkka kuvaus toimitusketjuista ja tuotteiden koostumuksista. Se taas voi olla haasteellista toimitusketjujen monimutkaisen rakenteen takia [Blackhurst *et al.*, 2005].

4.1. Jäljitettävyys

Moe [1998] määrittelee jäljitettävyyden (traceability) kykynä jäljittää tuotetta, sen historiaa, käyttöä tai sijaintia toimitusketjussa. Tuotteen jäljitettävyys tarkoittaa, että tuotteen valmistamiseen liittyvä tieto esimerkiksi tuotetta käsitelleistä toimijoista, tuotteen valmistuspaikasta sekä tuotteisiin kohdistuneista operaatioista voidaan tallentaa myöhempää käyttöä varten. Jäljitettävyys koskee koko tuotteen elinkaarta raaka-aineesta kuluttajalle ja mahdollisesti edelleen kierrätyksen kautta raaka-aineeksi. Tuotteella tarkoitetaan tässä mitä tahansa materiaalia missä tahansa valmistusprosessin vaiheessa [Moe, 1998]. Tuote voi lisäksi viitata joko yksittäiseen tavaraan, tavaraerään tai raaka-ainemassaan.

Handfieldin ja Nicholsin [1999] mukaan toimitusketju (supply chain) käsittää kaikki toiminnot, jotka liittyvät tuotteiden muuntumiseen ja siirtymiseen raaka-ainetasolta loppukäyttäjälle mukaan lukien tiedonkulun prosessien välillä. Usein toimitusketju määritellään laajemmin niin, että siihen lasketaan toimintojen ja tiedon lisäksi kuulumaan myös kaikki tuotteen käyttäjälle saattamiseen

liittyvät toimijat, laitteisto, logistiikka ja resurssit [Sahin and Robinson, 2002; Shapiro, 2009]. Näin toimitusketjuista muodostuu monimutkaisia verkostoja, joissa tuotteet ja informaatio siirtyvät toimijoiden ja toimintojen välillä.

Tässä työssä noudatetaan Handfieldin ja Nicholsin [1999] sekä Junkkarin ja Sirkan [2011b] mukaista rajatumpaa käsitystä toimitusketjusta. Toimitusketju määritellään toisiaan seuraavien prosessien joukkona, johon kuuluvat kaikki jonkin tuotteen valmistukseen liittyvät prosessit. Prosessien välillä kulkee materiaaleja ja tietoa, ja prosessin lopputuotetta käytetään raaka-aineena seuraavassa prosessissa. Jokaiseen prosessiin liittyy resursseja ja päästöjä, jotka kohdennetaan prosessien tuotteille.

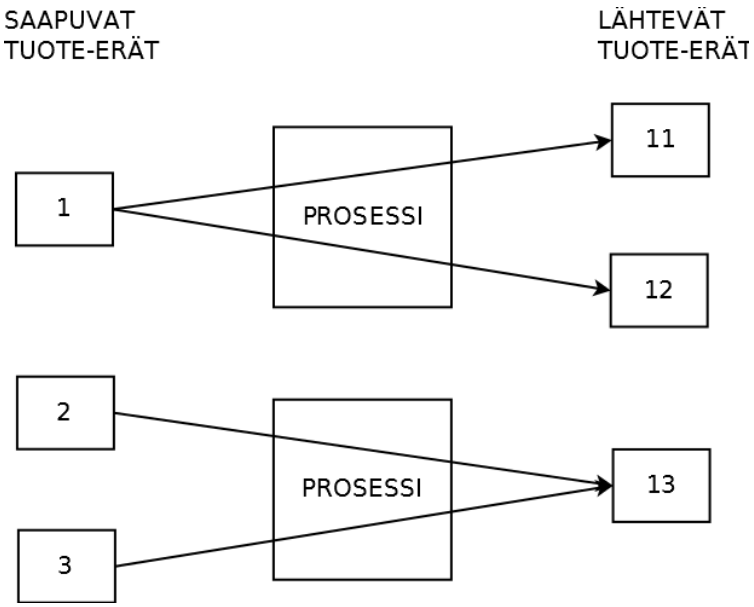
Tuotteiden jäljitettävyydessä olennaista on, että tuotteen historiaa voidaan jäljittää tuotantoketjussa sekä eteenpäin että taaksepäin. Eteenpäin jäljittämistä kutsutaan seurannaksi (tracking) ja taaksepäin seuraamista jäljittämiseksi (tracing). Sekä seurantaa että jäljittämistä tarvitaan tuotteen elinkaarta analysoitaessa.

Seuranta tarkoittaa tuotteen valmistuspolun seuraamista toimitusketjussa niin, että milloin tahansa voidaan tutkia, missä toimitusketjun prosesseissa tuote on ollut osallisena [Thompson *et al.*, 2005]. Seuranta on olennainen työkalu tilanteissa, joissa tuotteita joudutaan vetämään pois markkinoilta esimerkiksi sen jälkeen, kun toimitusketjussa on havaittu ongelma, joka on saattanut vaikuttaa tuotteiden laatuun.

Jäljittäminen viittaa tuotteen alkuperän ja ominaisuuksien selvittämiseen kulkemalla toimitusketjua taaksepäin [Bechini *et al.*, 2005]. Jäljittäminen liittyy esimerkiksi tuotteiden laatuongelmien selvittämiseen. Kun tiedetään viallisen tuotteen koko reitti, voidaan jäljitettävyysdatan avulla selvittää, mikä toimitusketjun vaihe ongelman aiheuttaa.

Jäljitettävyydestä puhuttaessa erotellaan usein koko toimitusketjun jäljitettävyys (chain traceability) ja prosessin sisäinen jäljitettävyys (internal traceability) [Senneset *et al.*, 2007]. Toimitusketjun jäljitettävyydellä tarkoitetaan tuotteiden seuraamista toimitusketjun eri vaiheiden välillä: toimitusketjun toimijoiden on tiedettävä käsittelemiensä tuotteiden reitti vähintään yksi askel eteenpäin ja taaksepäin toimitusketjussa [Senneset *et al.*, 2007]. Tämä edellyttää, että jäljitettävien tuote-erien identiteetit tallennetaan, kun tuote-eriä saapuu tai lähtee.

Sisäinen jäljitettävyys tarkoittaa linkitystä prosessiin saapuvien ja siitä lähtevien tuotteiden välillä [Senneset *et al.*, 2007]. Sisäinen jäljitettävyys edellyttää, että prosessissa tallennetaan siihen saapuvien tuote-erien identiteetit, määrätään yksilölliset identiteetit prosessissa luotaville uusille tuote-erille ja luodaan yhteys näiden identiteettien välille (kuva 3). Mitä tarkemmin sisäinen jäljitettävyys on toteutettu, sitä tarkemmin voidaan ongelmatilanteissa selvittää, mitä tuote-eriä ongelma koskee. Näin ollen esimerkiksi tuotteiden takaisinveto voidaan tarvittaessa toteuttaa rajatummin, kun tiedetään tarkasti, mitkä prosessista lähteneet tuote-erät sisältävät mitäkin tuotteita [Evira, 2011].



Kuva 3. Prosessin sisäinen jäljitettävyys.

Jäljitettävän tuote-erän tai yksikön tarkkuus usein vaihtelee toimitusketjun vaiheesta riippuen [Bechini *et al.*, 2008]. Etenkin elintarvikkeiden tuotannossa tarkkojen tietojen kerääminen on vaikeampaa tuotantoketjun alussa, koska luonnosta kerättävälle tuotteelle, kuten hedelmille tai maidolle, on huomattavasti vaikeampaa määrittää yksiselitteistä identiteettiä kuin esimerkiksi purkitetulle hedelmäsoseelle tai maitopurkille. Tuotantoketjun alkupäässä jäljitettävä tuote-erä voikin olla esimerkiksi samana päivänä (samoissa olosuhteissa) kerätty raaka-aine-erä.

4.2. Toimitusketjujen mallintaminen

Jäljitettävyysjärjestelmät koostuvat tallennustapahtumista, jotka sisältävät tuotteen tai ainesosan koko reitin toimittajilta markkinoille [Bechini *et al.*, 2005]. Järjestelmän täytyisi pystyä tunnistamaan jokainen tuote yksilöllisesti ja kerätä tietoa jokaisesta tuotteeseen kohdistuneesta toimenpiteestä, kuten kuljetuksesta tai prosesseista, joissa tuotetta muunnellaan tai joissa siihen kohdistuu toimintoja. Jäljitettävyysjärjestelmän tärkein ominaisuus onkin kyky käsitellä sekä tuotteita että toimintoja [Kim *et al.*, 1995; Jansen-Vullers *et al.*, 2003].

Koska jäljitettävyysjärjestelmien tavoitteena on, että valmiin tuotteen reitti ja tuotteen valmistamiseen käytetyt raaka-aineet ja komponentit voidaan selvittää jälkikäteen, yksi jäljitettävyyden perusedellytyksistä on tuotteiden yksiselitteinen tunnistaminen [Senneset *et al.*, 2007]. Kukin raaka-aine- tai tuote-erä on pystyttävä tunnistamaan, ja erien keskinäiset suhteet on tallennettava. Jokaisesta tuote-erästä on siis tiedettävä, mitä osatuotteita sen valmistuksessa on käytetty, jotta lopputuotteen koostumus voidaan selvittää tarkasti. Toisaalta keskinäisten suhteiden tallentaminen on tärkeää, jotta voidaan selvittää ne lopputuotteet, joiden valmistuksessa tiettyä tuote-erää on käytetty. [Jansen-Vullers *et al.*, 2003]

Jansen-Vullers ja muut [2003] erottelevat tekemänsä tapaustutkimuksen perusteella neljä vaatimusta, jotka jäljitettävyysmallin täytyy toteuttaa. Eri teollisuuden toimialoilta kerättyjen piirteiden pohjalta muodostettujen vaatimusten mukaan mallilta vaaditaan tietyt ominaisuudet, jotta

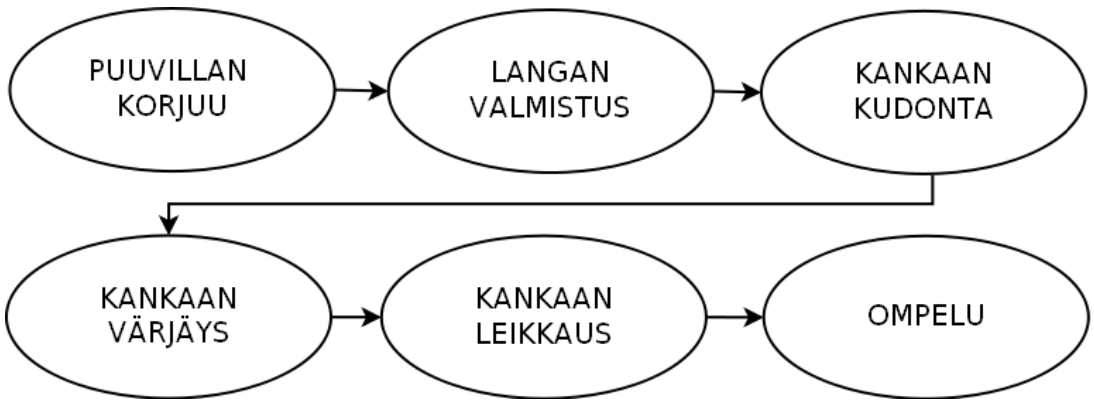
se on pätevä ja yleistettävissä mihin tahansa tuotantoketjun. Nämä ominaisuudet ovat kyky tallentaa tietoa tuotteiden keskinäisistä suhteista, tuotteisiin kohdistuneista toiminnoista, toimintoihin liittyvistä muuttujista ja arvoista sekä käytetyistä tuotantovälineistä. Jos malli täyttää nämä neljä vaatimusta, sen pitäisi olla yleistettävissä mihin tahansa jäljitettävyystapaukseen.

Moe [1998] puolestaan jakaa jäljitettävyysjärjestelmän kahteen osaan: tuotteiden reitteihin ja jäljitettävyyden kattavuuteen. Reitit kuvaavat tuotteiden kulkua ja tunnistamista tuotantoketjussa, ja kattavuus tarkoittaa, kuinka tarkkoja tietoja tuotteista ja toiminnoista tallennetaan. Kuten aiemmin todettiin, jäljitettävyyden ydinkäsitteet ovat tuotteet ja toiminnot, mutta järjestelmästä riippuu, kuinka laajasti niihin liittyviä tietoja tallennetaan. Esimerkiksi valmistusprosessista on tiedettävä vähintään sen tyyppi ja tapahtuma-aika, mutta prosessin kuvausta voidaan laajentaa sisällyttämällä siihen esimerkiksi tiedot prosessin ympäristökuormituksesta. [Moe, 1998]

Toimitusketjuja mallinnetaan usein verkkopohjaisilla menetelmillä. Blackhurst ja muut [2005] luettelevat useita syitä verkkomallien suosiolle. Ensinnäkin verkkomallit voidaan johtaa suoraan toimitusketjun rakenteesta, joten ne ovat helposti tulkittavissa. Niillä saadaan talteen osittain samanaikaisestikin tapahtuvien, toisistaan riippuvien toimintojen rakenne ja järjestyssuhteet. Lisäksi verkkomaiset mallit ovat visuaalisuutensa ansiosta havainnollistavia esityksiä monimutkaisista järjestelmistä. Usein käytettyjä verkkopohjaisia menetelmiä ovat esimerkiksi Petri-verkot ja niiden laajennukset [Zurawski and Zhou, 1994; Yu *et al.*, 2003].

4.3. Esimerkkisovellus

Tässä työssä käytetään esimerkkinä yksinkertaistettua tapausta T-paidan valmistuksesta. Valmistusprosessista esitetään kuusi vaihetta, jotka on esitetty kuvassa 4. Prosessin ensimmäinen vaihe on puuvillan korjuu, jonka lopputuote on puuvillakuitu. Puuvillakuitua käytetään langan valmistuksessa. Lanka menee seuraavaksi kudontaan, jossa valmistuu puuvillakangasta. Puuvillakangas värjätään, minkä jälkeen värjätty kangas leikataan, ja lopuksi leikatut kappaleet ommellaan yhteen T-paidaksi.



Kuva 4. Esimerkkitapauksen valmistusprosessi.

4.4. Jäljitettävyysgraafi

Jäljitettävyysgraafi on datakeskeinen työnkulkumalli fyysisten tuotteiden toimitusprosessien mallintamiseen. Sillä mallinnetaan tuotteiden valmistusprosesseja sekä prosesseihin liittyvien raaka-aineiden ja informaation virtaa. Jäljitettävyysgraafin avulla voidaan myös kohdentaa prosessiin liittyviä ominaisuuksia, kuten siinä aiheutuneita kustannuksia, prosessissa käsitellyille tuotteille. [Junkkari and Sirkka, 2011a; Junkkari and Sirkka, 2014]

Jäljitettävyysgraafi koostuu solmuista, kaarista ja näiden ominaisuuksista. Solmut kuvaavat prosesseja ja kaaret informaation ja tuotteiden siirtymistä prosessien välillä.

Jäljitettävyysgraafissa solmu edustaa prosessia, jossa tarvitaan jotain resursseja tai jossa syntyy uusia kustannuksia. Prosessisolmulla on aina tunniste, tyyppi sekä joukko tuote-eriä ja attribuutteja. Prosessit voidaan jaotella prosessityypeittäin niin, että samanlaiset prosessit ovat saman prosessityypin edustajia. Prosessit ovat keskenään samanlaisia, jos niiden lopputuotteet ovat keskenään samanlaiset.

4.4.1. Jäljitettävyysgraafin primitiivit

Jäljitettävyysgraafin solmuilla ja kaarilla on omat primitiivinsä. Solmuihin liittyvät primitiivit ovat objekti, tuote-erä ja attribuutti. Kaarien primitiivit ovat siirtyvä tuote-erä, objektikuvaus ja johdettu attribuutti.

Objekti (object) on toimitusketjussa yksilöllisesti tunnistettavissa oleva tuote. Objekti voidaan jäljitettävyysjärjestelmän tarkkuudesta riippuen rinnastaa yksittäiseen tuotteeseen, tuote-erään tai materiaalimassaan.

Tuote-erä (product portion) määrittelee prosessisolmuun liittyvän tuotteen ja tuotteelle määräytyvän osuuden prosessin kustannuksista. Tuote-erä sisältää suhdeluvun, joka ilmaisee, kuinka suuri osuus kyseiseen solmuun liittyvistä kustannuksista kohdennetaan tuote-erän tuotteelle.

Attribuutit kuvaavat prosessin ominaisuuksia. Syöteattribuutit (input attributes) ovat prosessiin liittyviä kustannuksia, kuten raaka-aineita ja resursseja. Tulosattribuutit (output attributes) kuvaavat prosessissa varsinaisten tuotteiden lisäksi syntyviä aineita, kuten päästöjä ja jätteitä. Kaikki muu prosessiin liittyvä tieto ilmaistaan informaatioattribuuteilla (info attributes). Jokaiseen attribuuttiin liittyy kaksi arvoa: varsinainen prosessiin liittyvä arvo ja kumuloitunut arvo, joka lasketaan prosessia edeltävien prosessien arvoista.

Jäljitettävyysgraafissa solmu sisältää aina tunnisteen sekä joukon tuote-eriä ja attribuutteja. Kaaren tunnisteena ovat sen alku- ja loppusolmu, minkä lisäksi kaareen liittyy joukko primitiivejä: siirtyvä tuote-erä, objektikuvaus ja johdettu attribuutti.

Siirtyvä tuote-erä (sifted product portion) tarkoittaa tuotteita, jotka siirretään prosessista toiseen. Siirtyvä tuote-erä saattaa sisältää vain osan alkuperäisen tuote-erän objekteista. Siirtyvään tuote-erään liittyy suhdeluku, joka kertoo, kuinka suuri osa edellisen solmun objekteista siirtyvään tuote-erään liittyy.

Objektikuvaus (object mapping) liittyy siirtyvään tuote-erään. Objektikuvaus voi olla ekvivalenssi (equivalence), osajoukko (subsetting), ylijoukko (supersetting), jakaminen (division) tai kooste (composition). Ekvivalenssissa alku- ja loppusolmun objektit ovat samat. Osajoukko tarkoittaa, että osa alkusolmun objekteista siirtyy loppusolmuun. Ylijoukko tarkoittaa, että kaikki alkusolmun objektit siirretään loppusolmuun, minkä lisäksi loppusolmuun saapuu objekteja muistakin prosesseista. Koosteessa useista objekteista kootaan yksi objekti, toisin sanoen alkusolmun tuotteita käytetään loppusolmun tuotteiden komponentteina. Jakaminen tarkoittaa, että yksittäinen objekti jaetaan loppusolmussa useaksi objektiksi.

Objektikuvauksissa objektien identiteetit muuttuvat koosteessa ja jakamisessa. Objektit voivat muuttua toisiksi objekteiksi sekä loogisella tasolla (tietokannassa) että fyysisellä tasolla (tosimaailmassa). Tämä objektien muuntuminen on otettava huomioon myös työnkulkumallissa.

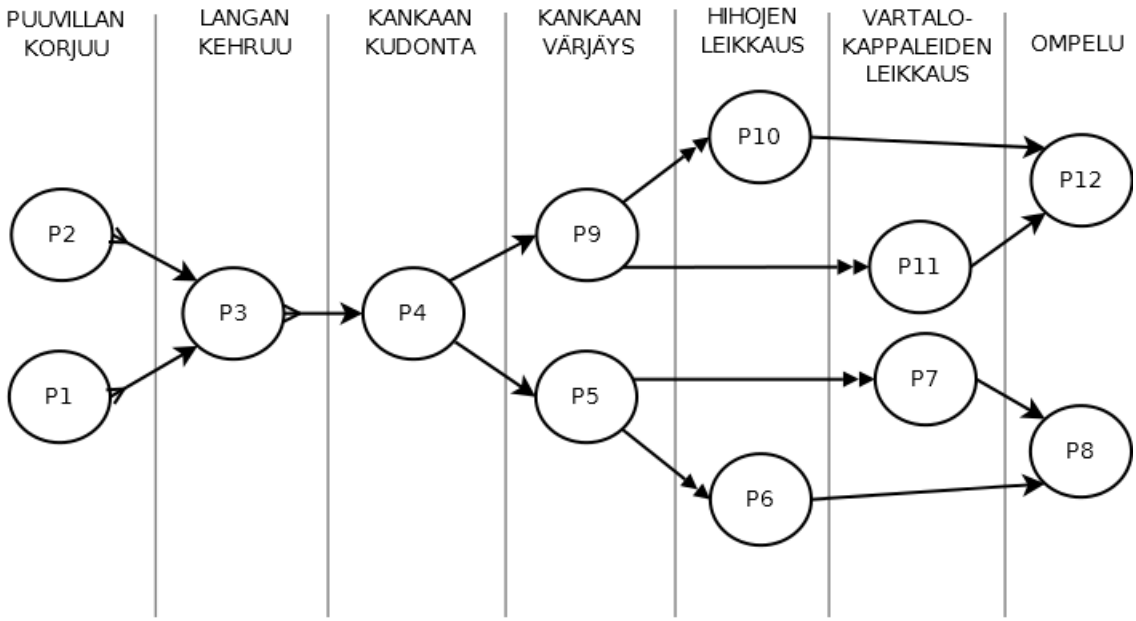
Johdettu attribuutti kertoo siirtyvään tuote-erään liittyvien päästöjen ja raaka-aineiden määrän. Se lasketaan käyttämällä prosessikohtaisia sääntöjä. [Junkkari and Sirkka, 2011a]

4.4.2. Jäljitettävyysgraafin graafinen esitys

Jäljitettävyysgraafin graafinen esitys on yleisen tason kuvaus koko valmistusprosessista. Esityksessä kuvataan informaation siirtymistä ja objektien muuntumista prosessien välillä. Graafisessa esityksessä noudatetaan perinteistä tapaa esittää graafi: solmut kuvataan ympyröinä ja kaaret nuolina. Nuolia varten käytetään kolmea eri merkintätapaa:

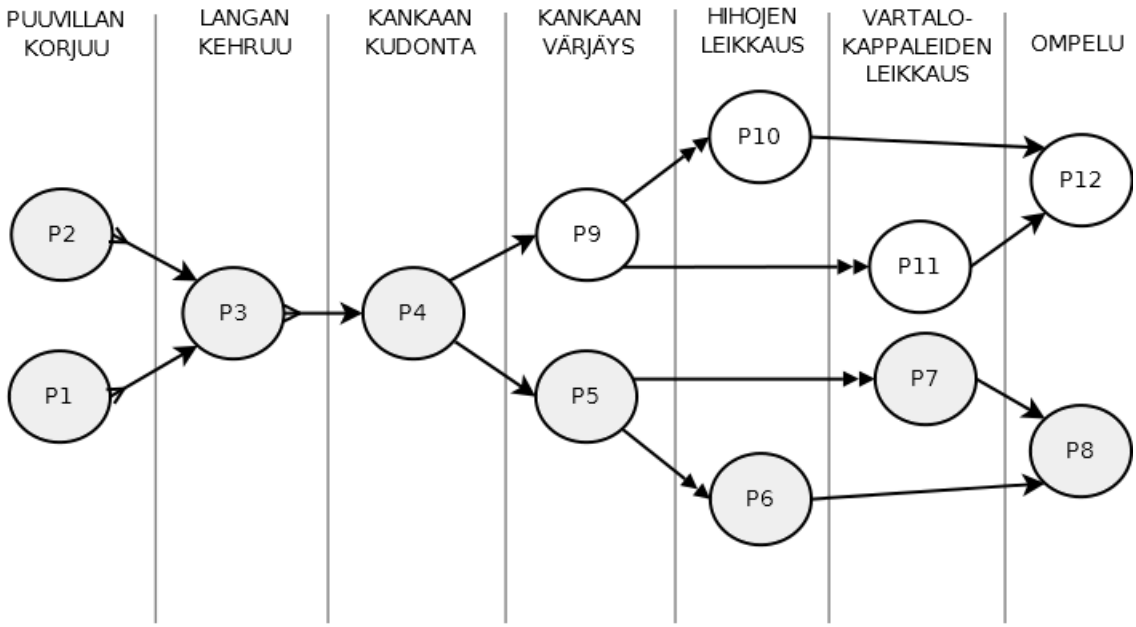
- Tavallinen kaari kuvaa tilannetta, jossa alkusolmusta lähteneet objektit säilyttävät identiteettinsä loppusolmussa. Tavallinen kaari esitetään tavallisena nuolena.
- Jakava kaari kuvaa alkusolmun yksittäisen objektin jakamisen useaksi objektiksi loppusolmussa. Jakava kaari esitetään kaksoiskärjellisenä nuolena.
- Kokoava kaari kuvaa tilannetta, jossa useasta alkusolmun objektista muodostetaan loppusolmussa yksi objekti. Kokoava kaari kuvataan alkupäästä haarautuneella nuolella.

Kuvassa 5 on aiemmin esitellystä T-paidan valmistusprosessista muodostettu jäljitettävyysgraafi. Tuotteet siirtyvät graafissa vasemmalta oikealle. Esimerkiksi puuvillan korjuussa syntyvät puuvillapaalit siirtyvät langan kehruuseen, jossa useasta paalista kehrätään yksi lankarulla. Solmujen välillä on siten kokoava kaari, koska kaareen liittyvä objektikuvaus on kooste. Kankaan värjäyksen ja kangaskappaleiden leikkauksen välissä on puolestaan jakava kaari, koska leikkauksessa yksi kangaspakka jaetaan useaksi kangaskappaleeksi.



Kuva 5. Jäljitettävyysgraafi.

Toimitusketju on jäljitettävyysgraafin aligraafi. Kuvassa 6 esitetään mahdollinen toimitusketju kuvan 5 jäljitettävyysgraafin solmussa P8 syntyvälle T-paidalle.



Kuva 6. Mahdollinen toimitusketju solmussa P8 valmistettavalle tuotteelle. Toimitusketjun solmut on merkitty kuvaan harmaalla.

4.4.3. Esimerkki jäljitettävyysgraafista

Tässä luvussa annetaan kuvaan 6 perustuva esimerkki jäljitettävyysgraafista. Solmujen ja kaarien esittämisessä käytetään Junkkarin ja Sirkan [2011b] käyttämää merkintätapaa, jossa solmut ja kaaret esitetään kulmasulkeilla merkittyinä monikkoina. Joukot on merkitty aaltosulkeilla.

Esimerkkigraafissa käydään läpi kuvaan 6 harmaalla merkityistä solmuista P1–P8 muodostuva toimitusketju. Sen solmuihin viitataan tekstissä vain solmun numerolla. Objektien painoina, attribuuttien arvoina ja suhdelukuina käytettävät arvot ovat keksittyjä esimerkkejä, jotka eivät perustu todellisiin arvoihin.

Solmu 1 on korjuusolmu, johon liittyy kolme tuote-erää. Tuote-erä on monikko ⟨N, V, ID-joukko, R⟩, jossa N on tuotteen nimi, V on tuote-erän koko (esimerkiksi paino tai tilavuus), ID-joukko on joukko tuote-erään kuuluvia objekti-identiteettejä ja R on suhdeluku, joka kertoo tuote-erän osuuden kaikista solmun tuotteista. Esimerkiksi solmun 1 ensimmäinen tuote-erä ⟨Puuvillapaali, 2 000 kg, {id1, …, id10}, 0,6⟩ tarkoittaa, että solmuun liittyy 2 000 kg:n edestä (10 kappaletta) puuvillapaaleja, joiden identiteetit ovat id1–id10. Suhdeluku 0,6 tarkoittaa, että 60 % solmuun liittyvistä kustannuksista kohdennetaan puuvillapaaleille. Puuvillansiementen ja korjuujätteen identiteettijoukot ovat tyhjät, koska niitä käsitellään massana, jolla ei ole identiteettiä.

Solmuun liittyy myös attribuutteja. Attribuutti on monikko ⟨N_A, T, A, K⟩, jossa N_A on attribuutin nimi, T attribuutin tyyppi (syöte-, tulos- tai informaatioattribuutti), A solmuun liittyvä attribuutin arvo ja K attribuutin kumuloitunut arvo. Esimerkiksi solmun 1 korjuuprosessissa syntyy 100 kg hiilidioksidipäästöjä (CO_2) ja kuluu 50 litraa dieseliä. Solmussa 1 attribuuttien perusarvo ja kumuloitunut arvo ovat samat, koska solmu on alkusolmu, jolla ei ole edeltäviä solmuja.

⟨1, Korjuu,
{⟨Puuvillapaali, 2 000 kg, {id1, …, id10}, 0,6⟩,
⟨Puuvillansiemen, 4 000 kg, ∅, 0,3⟩,
⟨Korjuujäte, 150 kg, ∅, 0,1⟩},
{⟨CO_2, tulos, 100 kg, 100 kg⟩,
⟨Diesel, syöte, 50 litraa, 50 litraa⟩,
⟨Yhtiö, info, {y1}, {y1}⟩,
⟨Sijainti, info, {lat 30.19 - lon 149.40}, {lat 30.19 - lon 149.40}⟩}⟩

Toisessa korjuusolmussa sama määrä puuvillapaaleja tuotetaan hieman pienemmillä kustannuksilla.

⟨2, Korjuu,
{⟨Puuvillapaali, 2 000 kg, {id11, …, id20}, 0,6⟩,
⟨Puuvillansiemen, 4 000 kg, ∅, 0,3⟩,
⟨Korjuujäte, 150 kg, ∅, 0,1⟩},
{⟨CO_2, tulos, 90 kg, 90 kg⟩,
⟨Diesel, syöte, 40 litraa, 40 litraa⟩,
⟨Yhtiö, info, {y2}, {y2}⟩,
⟨Sijainti, info, {lat 31.05 - lon 150.55}, { lat 31.05 - lon 150.55}⟩}⟩

Osa solmujen 1 ja 2 paaleista valitaan langan valmistukseen solmuun 3. Tätä kuvataan kaarella ⟨S_A, S_L, SP, J⟩, jossa S_A on kaaren alkusolmu, S_L kaaren loppusolmu, SP siirtyvä tuote-erä ja J joukko johdettuja attribuutteja. Esimerkkitapauksessa 50 % solmun 1 puuvillapaaleista valitaan langan kehruuseen. Koska puuvillapaalien osuus korjuun kustannuksista on 60 %, tulos- ja

syöteattribuuttien arvot kerrotaan kertoimilla 0,5 ja 0,6. Näin esimerkiksi CO_2:n johdetuksi arvoksi tulee $100 \cdot 0{,}6 \cdot 0{,}5 = 30$.

Siirtyvä tuote-erä on monikko ⟨PN, C, M, Rp⟩, jossa PN on tuotteen nimi, C tuote-erän koko, M objektien identiteetin muutokset esittävä objektikuvaus ja Rp siirtyvän tuote-erän suhdeluku. Suhdeluku kertoo siirtyvän tuote-erän osuuden kaikista edellisen solmun tuotteista. Alla olevat solmuun 3 päättyvät kaaret ovat kokoavia kaaria, koska kehruussa puuvillapaaleista koostetaan lankarullia. Identiteettien muutokset kuvataan siirtyvään tuote-erään liittyvässä objektikuvauksessa. Esimerkiksi merkintä ⟨{id1, id2}, id30⟩ tarkoittaa, että objektit id1 ja id2 (puuvillapaalit) muodostavat objektin id30 (lankarulla).

⟨1, 3, ⟨Puuvillapaali, 1 000 kg, {⟨{id1, id2}, id30⟩, ⟨{id3, id4}, id31⟩, ⟨id5, id32⟩}, 0,5⟩,
{⟨CO_2, tulos, 30 kg⟩,
⟨Diesel, syöte, 15 litraa⟩,
⟨Yhtiö, info, {y1}⟩}⟩

Solmun 2 paaleista vain yksi (10 %) siirtyy solmuun 3. Se yhdistetään solmusta 1 tulevan paalin kanssa lankarullaksi id32.

⟨2, 3, ⟨Puuvillapaali, 200 kg, {⟨id11, id32⟩}, 0,1⟩,
{⟨CO_2, tulos, 5,4 kg⟩,
⟨Diesel, syöte, 2,4 litraa⟩,
⟨Yhtiö, info, {y2}⟩}⟩

Solmussa 3 puuvilla kehrätään langaksi. Solmun attribuuteista CO_2:lla ja dieselillä on edellisistä solmuista kumuloituneet arvot. Attribuutin kumuloitunut arvo lasketaan laskemalla yhteen solmuun saapuvien kaarien johdetut arvot ja attribuutin perusarvo. Näin esimerkiksi CO_2:n kumuloitunut arvo solmussa 3 on 30 kg + 5,4 kg + 10 kg = 45,4 kg. Informaatioattribuuttien arvot siirtyvät sellaisenaan, joten Yhtiö-attribuutin arvoina ovat kahdesta korjuuprosessista vastanneet yhtiöt sekä nykyisestä kehruuprosessista vastaava yhtiö. Sähkö on uusi attribuutti, jonka perusarvo ja kumuloitunut arvo ovat toistaiseksi samat.

⟨3, Kehruu,
{⟨Lanka, 1 000 kg, {id30, id31, id32}, 0,9⟩,
⟨Kehruujäte, 200 kg, ∅, 0,1⟩},
{⟨CO_2, tulos, 10 kg, 45,4 kg⟩,
⟨Diesel, syöte, 0 litraa, 17,4 litraa⟩,
⟨Sähkö, syöte, 25 kWh, 25 kWh⟩,
⟨Yhtiö, info, {y3}, {y1, y2, y3}⟩}⟩

Solmusta 3 kaikki lanka siirtyy kudottavaksi solmuun 4, joten seuraavan kaaren siirtosuhde on 1. Langan osuus edellisen solmun kustannuksista on 90 %, joten syöte- ja tulosattribuuttien arvot

kerrotaan 1 · 0,9:llä. Kyseessä on jälleen kokoava kaari, koska kudonnassa lankarullista koostetaan kangaspakkoja. Lankarullista id30 ja id31 muodostetaan pakka, jonka tunnus on id40, ja rullasta id32 pakka, jonka tunnus on id41.

⟨3, 4, ⟨Lanka, 1 000 kg, {⟨{id30, id31}, id40⟩, ⟨{id32}, id41⟩}, 1⟩,
{⟨CO_2, tulos, 40,86 kg⟩,
⟨Diesel, syöte, 15,66 litraa⟩,
⟨Sähkö, syöte, 22,5 kWh⟩,
⟨Yhtiö, info, {y1, y2, y3}⟩}⟩

Alla oleva solmu 4 kuvaa kankaan kudontaa. Kudonnassa kankaan osuudeksi lasketaan 100 % prosessin kustannuksista.

⟨4, Kudonta,
{⟨Kangas, 950 kg, {id40, id41}, 1⟩,
⟨Kangasjäte, 50 kg, ∅, 0⟩},
{⟨CO2, tulos, 5 kg, 45,86 kg⟩,
⟨Diesel, syöte, 0 litraa, 15,66 litraa⟩,
⟨Sähkö, syöte, 20 kWh, 45,5 kWh⟩,
⟨Yhtiö, info, {y4}, {y1, y2, y3, y4}⟩}⟩

Kudonnasta kaikki kangas siirtyy värjättäväksi. Kangaspakkojen identiteetit muuttuvat: kangaspakat pysyvät pientä hävikkiä lukuun ottamatta sinänsä samoina, mutta niiden katsotaan olevan eri tuote värjäyksen jälkeen. Attribuuttien arvot eivät tällä kertaa muutu, koska siirtyvän tuote-erän suhdeluku on 1, samoin kuin tuote-erän osuus edellisen solmun kustannuksista.

⟨4, 5, ⟨Kangas, 950 kg, {⟨id40, id42⟩, ⟨id41, id43⟩}, 1⟩,
{⟨CO_2, tulos, 45,86 kg⟩,
⟨Diesel, syöte, 15,66 litraa⟩,
⟨Sähkö, syöte, 45,5 kWh⟩,
⟨Yhtiö, info, {y1, y2, y3, y4}⟩}⟩

Värjäyksessä kankaaseen kohdistetaan 95 % prosessin kustannuksista. Kudonnasta ja värjäyksestä vastaa sama yhtiö, joten informaatioattribuutin kumuloitunut arvo ei muutu.

⟨5, Värjäys,
{⟨Värjätty kangas, 900 kg, {id42, id43}, 0,95⟩,
⟨Kangasjäte, 50 kg, ∅, 0,05⟩},
{⟨CO_2, tulos, 20 kg, 65,86 kg⟩,
⟨Diesel, syöte, 0 litraa, 15,66 litraa⟩,
⟨Sähkö, syöte, 30 kWh, 75,5 kWh⟩,
⟨Vesi, syöte, 50 000 litraa, 50 000 litraa⟩,
⟨Yhtiö, info, {y4}, {y1, y2, y3, y4}⟩}⟩

Värjätystä kankaasta leikataan kahdenlaisia kappaleita, joten värjätyt kankaat siirtyvät kahteen eri leikkausprosessiin: hihakappaleiden leikkaukseen (solmu 6) ja vartalokappaleiden leikkaukseen (solmu 7). Esimerkkitapauksessa kangaspakasta id42 leikataan 4 000 T-paidan vartalokappaletta ja pakasta id43 leikataan 4 000 T-paidan hihakappaletta. Tämä identiteettimuutos kuvataan myös kaarien objektikuvauksissa. Seuraavat kaaret ja solmut kuvaavat kankaan leikkausta paloiksi.

⟨5, 6, ⟨Värjätty kangas, 300 kg, {⟨id43, {id4051, …, id8051}⟩}, 0,2⟩,
{⟨CO_2, tulos, 12,513 kg⟩,
⟨Diesel, syöte, 2,975 litraa⟩,
⟨Sähkö, syöte, 14,345 kWh⟩,
⟨Vesi, syöte, 9 500 litraa⟩,
⟨Yhtiö, info, {y1, y2, y3, y4}⟩}⟩

⟨6, Hihojen leikkaus,
{⟨Hihakappale, 4 000 kpl, {id4051, …, id8051}, 0,9⟩,
⟨Leikkuujäte, 50 kg, ∅, 0,10⟩},
{⟨CO_2, tulos, 5 kg, 17,513 kg⟩,
⟨Diesel, syöte, 0 litraa, 2,975 litraa⟩,
⟨Sähkö, syöte, 10 kWh, 24,345 kWh⟩,
⟨Vesi, syöte, 0 litraa, 9 500 litraa⟩,
⟨Yhtiö, info, {y4}, {y1, y2, y3, y4}⟩}⟩

⟨5, 7, ⟨Värjätty kangas, 600 kg, {⟨id42, {id50, …, id4050}⟩}, 0,8⟩,
{⟨CO_2, tulos, 50,054 kg⟩,
⟨Diesel, syöte, 11,902 litraa⟩,
⟨Sähkö, syöte, 57,38 kWh⟩,
⟨Vesi, syöte, 38 000 litraa⟩,
⟨Yhtiö, info, {y1, y2, y3, y4}⟩}⟩

⟨7, Vartalokappaleiden leikkaus,
{⟨Vartalokappale, 4 000 kpl, {id50, …, id4050}, 0,9⟩,
⟨Leikkuujäte, 20 kg, ∅, 0,10⟩,
{⟨CO_2, tulos, 6 kg, 56,054 kg⟩,
⟨Diesel, syöte, 0 litraa, 11,902 litraa⟩,
⟨Sähkö, syöte, 12 kWh, 69,38 kWh⟩,
⟨Vesi, syöte, 0 litraa, 38 000 litraa⟩,
⟨Yhtiö, info, {y4}, {y1, y2, y3, y4}⟩}⟩

Lopuksi kangaskappaleet siirtyvät ommeltavaksi. Tässä oletetaan, että 10 % molemman tyyppisistä kappaleista ei esimerkiksi huonon laadun takia kelpaa ommeltavaksi. Näin ollen attribuuttien arvot kerrotaan suhteilla 0,9 ja 0,9 (hihakappaleiden osuus leikkaussolmun 6 kustannuksista). Alla olevat kaaret ovat jälleen kokoavia kaaria, koska ompelusolmussa kahdesta hihakappaleesta ja kahdesta vartalokappaleesta ommellaan yksi T-paita.

⟨6, 8, ⟨Hihakappale, 3 600 kpl,
{⟨{id4051, id4052}, id9000⟩, ⟨{id4053, id4054}, id9001⟩, …}, 0,9⟩,
{⟨CO_2, tulos, 14,186 kg⟩,
⟨Diesel, syöte, 2,10 litraa⟩,
⟨Sähkö, syöte, 19,72 kWh⟩,
⟨Vesi, syöte, 7 695 litraa⟩,
⟨Yhtiö, info, {y1, y2, y3, y4}⟩}⟩

⟨7, 8, ⟨Vartalokappale, 3 600 kpl,
{⟨{id50, id51}, id9000⟩, ⟨{id52, id53}, id9001⟩, …}, 0,9⟩,
{⟨CO_2, tulos, 45,404 kg⟩,
⟨Diesel, syöte, 9,641 litraa⟩,
⟨Sähkö, syöte, 56,20 kWh⟩,
⟨Vesi, syöte, 30 780 litraa⟩,
⟨Yhtiö, info, {y1, y2, y3, y4}⟩}⟩

Ompelusolmussa hiha- ja vartalokappaleet ommellaan yhteen. Solmun 8 syöte- ja tulosattribuuttien arvot on laskettu laskemalla yhteen solmuun saapuvien kaarien johdettujen attribuuttien arvot ja nykyisen solmun arvo, jolloin esimerkiksi sähkönkulutuksen arvoksi saadaan 19,72 kWh + 56,20 kWh + 10 kWh = 85,92 kWh.

⟨8, Ompelu, {⟨T-paita, 1 800 kpl, {id9000, …, id10800}, 0,98⟩,
⟨Ompelujäte, 10 kg, ∅, 0,02⟩,
{⟨CO_2, tulos, 5 kg, 64,59 kg⟩,
⟨Diesel, syöte, 0 litraa, 11,741 litraa⟩,
⟨Sähkö, syöte, 10 kWh, 85,92 kWh⟩,
⟨Vesi, syöte, 0 litraa, 38 475 litraa⟩,
⟨Yhtiö, info, {y5}, {y1, y2, y3, y4, y5}⟩}⟩

Ompelusolmun tietojen perusteella voidaan nyt laskea yhden T-paidan valmistukseen käytettyjen resurssien määrä. Laskennassa attribuuttien kumuloituneet arvot kerrotaan T-paidan osuudella solmun kustannuksista (0,98) ja yhden T-paidan osuudella kaikista tuote-erän T-paitaobjekteista (1 / 1 800). Esimerkiksi yhden T-paidan hiilijalanjäljeksi saadaan näin $(1 / 1\,800) \cdot 0{,}98 \cdot 64{,}59$ kg = 0,035 kg.

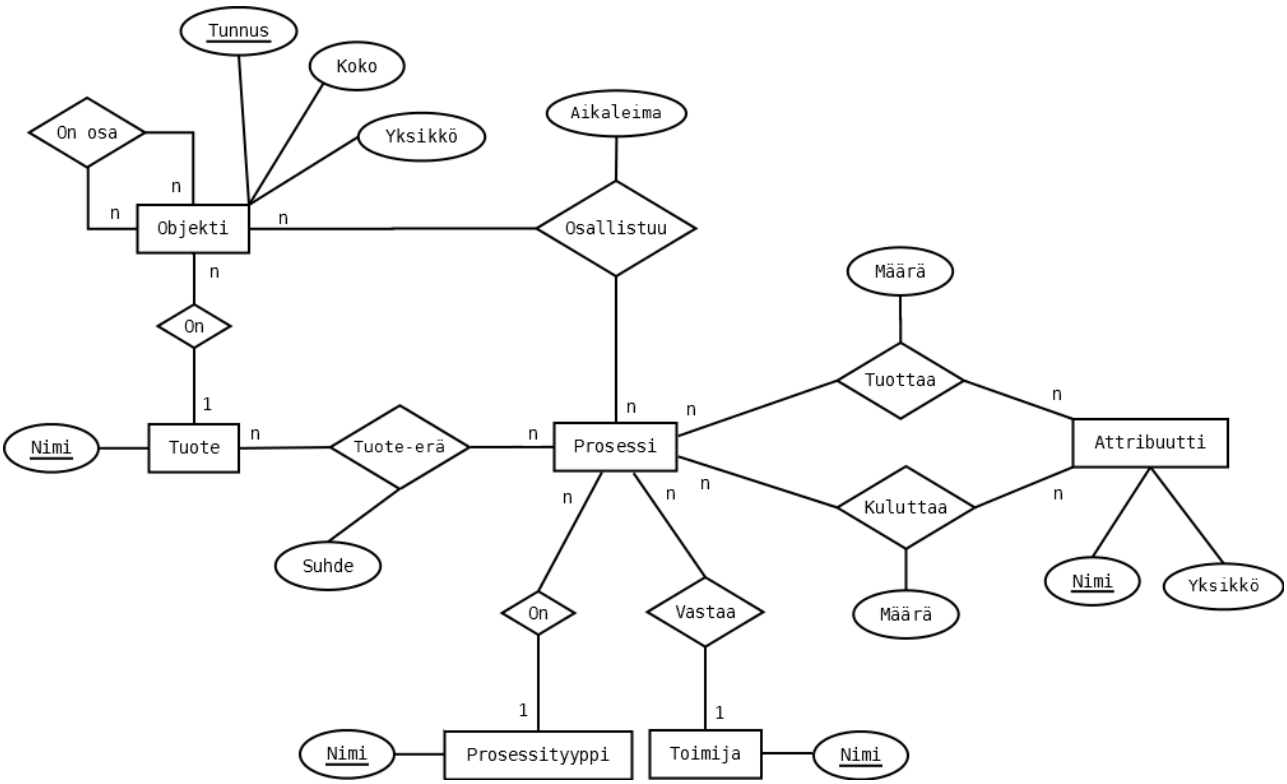
5. Jäljitettävyysgraafin toteutus graafitietokannassa

Junkkari ja Sirkka [2011a] ovat esittäneet jäljitettävyysgraafin toteutuksen relaatiotietokannassa. Tässä työssä jäljitettävyysgraafi on toteutettu graafitietokannassa, koska jäljitettävyysgraafi itsessään on graafi ja se on siten luontevaa tallentaa graafimuotoon. Koska graafitietokannat mahdollistavat suoraan graafille ominaiset operaatiot, niiden pitäisi sopia hyvin Junkkarin ja Sirkan [2011a] esittämille toimitusketjuun liittyville kyselyille. Jäljitettävyysgraafin tapauksessa tällaisia operaatiota ovat esimerkiksi objektin reitin selvittäminen graafissa ja objektia ja sen komponentteja käsitelleiden toimijoiden hakeminen.

Tässä luvussa esitellään jäljitettävyysgraafin toteutus Neo4j-graafitietokannassa. Kohdassa 5.1 kerrotaan yleisemmin jäljitettävyysgraafin sovittamisesta graafitietokantaan, ja kohdassa 5.2 esitellään tarkemmin tietokannan solmutyypit ja niiden väliset suhteet. Kohdassa 5.3 esitellään aiemmin kuvatun esimerkkitapauksen avulla, miten jäljitettävyysgraafi voidaan esittää graafitietokannassa.

5.1. Esimerkkitietokannan kuvaus

Kuvassa 7 on ER-mallin mukainen kaavio, jossa näkyy, millaisiksi olioiksi ja suhteiksi jäljitettävyysgraafi on jaettu tietokantaa suunniteltaessa.

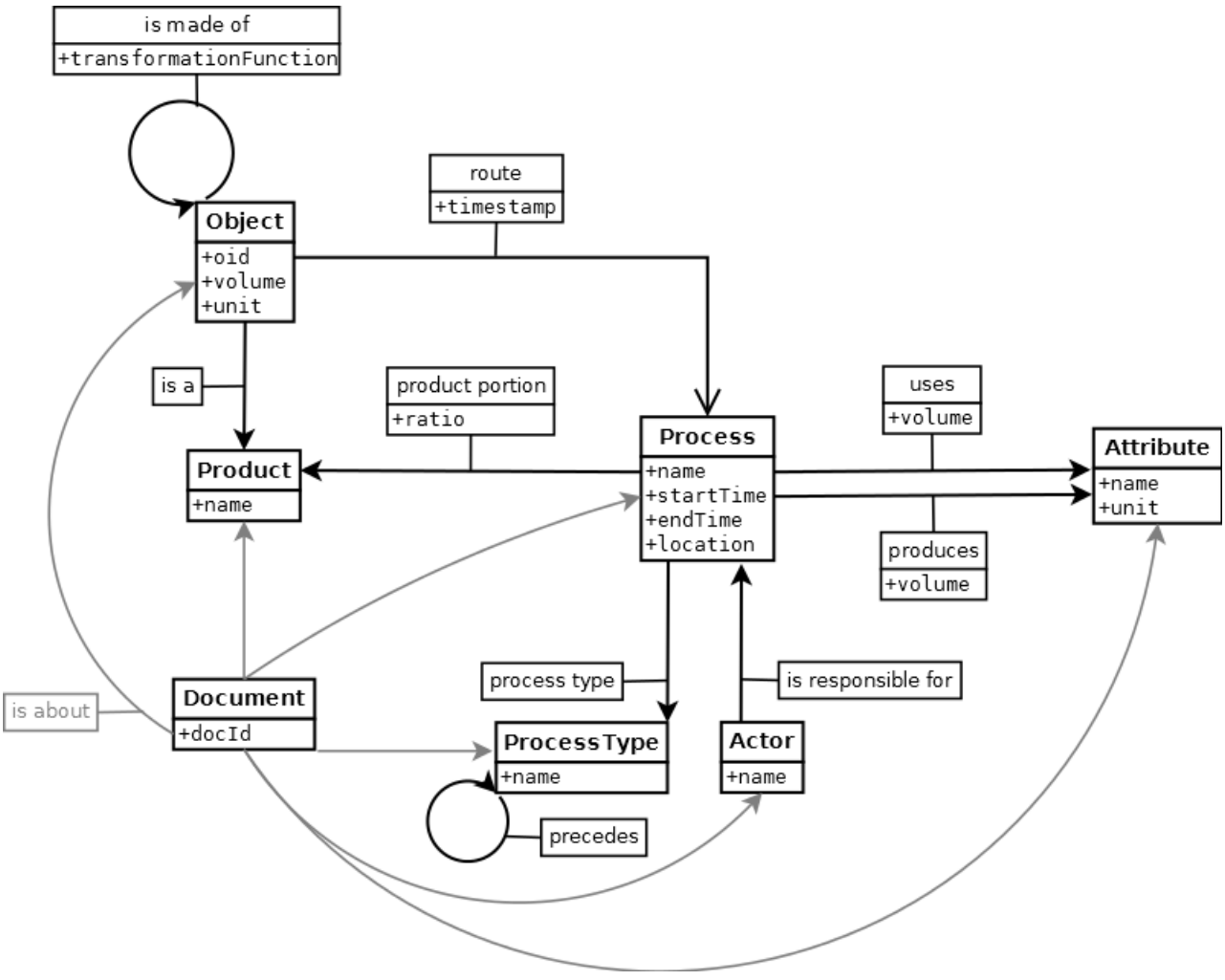


Kuva 7. ER-kaavio jäljitettävyysgraafista. Suorakulmiot kuvaavat olioita, vinoneliöt suhteita ja soikiot attribuutteja.

Jäljitettävyysgraafiin liittyvät oliot ovat objekti, tuote, prosessi, prosessityyppi, attribuutti ja toimija. Graafitietokantaa laadittaessa tietokantakaavio on helppoa johtaa ER-kaaviosta: oliot ovat

solmuja, suhteet suhteita ja attribuutit ominaisuuksia. Jäljitettävyysgraafin primitiiveistä objekti ja syöte- ja tulosattribuutti on mallinnettu tietokannassa solmuina. Informaatioattribuutit esitetään prosessisolmuihin liittyvinä ominaisuuksina, paitsi prosesseihin liittyvät toimijat ja dokumentit, jotka on mallinnettu solmuina. Toimijat on nostettu omaksi solmutyypikseen, koska ne ovat itsenäisiä olioita, joilla on ominaisuuksia ja suhteita muihin olioihin (sama toimija voi esimerkiksi liittyä moneen eri prosessiin). Samoin dokumentit, jotka jäljitettävyysgraafissa esitetään informaatioattribuutteina, mallinnetaan solmuina.

Esimerkkitietokannan rakenne on esitetty kuvassa 8. Kuvassa näkyvät tietokannan solmut ja niiden väliset suhteet sekä solmujen ja suhteiden ominaisuudet. Solmut on kuvattu suorakulmioina, jonka yläosassa on solmun tyyppi ja alaosassa sen ominaisuudet. Suhteet on kuvattu solmujen välisinä nuolina, joiden suunta osoittaa suhteen alku- ja loppusolmun. Suhteiden ominaisuudet on kuvattu nuoliin liittyvinä suorakulmioina, joiden yläosassa on suhteen tyyppi ja alaosassa sen ominaisuudet. Sekä suhteilla että solmuilla voi olla mitä tahansa muitakin ominaisuuksia kuin tietokantakaavioon merkityt, mutta kaavioon on esimerkin vuoksi merkitty yleisimmät tämän työn esimerkkitietokannassa käytetyt. Tietokannan solmuista ja suhteista kerrotaan tarkemmin seuraavassa alakohdassa.



Kuva 8. Tietokantakaavio.

Esimerkkitietokannassa suhteisiin tallennetaan ominaisuuksina numeerisia ja päivämäärätietoja. Esimerkiksi objektin ja prosessin välillä on ROUTE-suhde, joka kertoo, että objekti on kulkenut kyseisen prosessin kautta. ROUTE-suhteelle määritelty ”time”-ominaisuus puolestaan on aikaleima, joka kertoo, milloin objekti oli prosessissa.

Tunnuksia on lisätty tietokannassa neljälle solmutyypille: toimijoille (ACTOR), tuotteille (PRODUCT), attribuuteille (ATTRIBUTE) ja prosessityypeille (PROCESS_TYPE). Tunnuksia käytetään tietokannassa solmujen ryhmittelyyn, jotta tarvittaessa voidaan hakea helposti kaikki tietyntyyppiset solmut. Kaikille solmutyypeille ei ole lisätty tunnuksia, koska niitä ei tarvita tietokantaoperaatioissa. Tunnuksia ei myöskään ole merkitty tietokantakaavioon, koska niillä on käyttöä lähinnä tietokannan sisäisessä käytössä, eikä niillä ole merkitystä kaavion ymmärtämisen kannalta.

5.2. Esimerkkitietokannan solmu- ja suhdetyypit

Prosessityyppi edustaa toimitusketjun prosessia. Kukin vaihe toimitusketjussa edustaa eri toimintoa, ja prosessityyppi kuvaa näitä toimintoja. Kunkin prosessityypin lopputuote on eri tuote. Prosessityypit liittyvät tietokannassa prosesseihin ja itseensä: prosessityyppien välisellä PRECEDES-suhteella ilmaistaan prosessityyppien keskinäinen järjestys.

Prosessit ovat prosessityyppien esiintymiä. Ne liittyvät kaikkiin muihin tietokannan solmutyyppeihin. PROCESS_TYPE-suhde liittää prosessisolmun siihen prosessityyppiin, jonka ilmentymä prosessi on. Prosessiin tulevat ROUTE-suhteet ilmaisevat, mitkä objektit ovat osallistuneet prosessiin, eli esimerkiksi sen, mitä objektit kyseisessä prosessissa on valmistettu. Prosessin ja tuotteen välisellä PRODUCT_PORTION-suhteella ilmaistaan, miten prosessin kustannukset jakautuvat eri tuotteiden kesken. Se vastaa jäljitettävyysgraafin tuote-erää.

Prosessissa käytettävät resurssit ja siinä tuotettavat päästöt ilmaistaan liittämällä prosessisolmuun joukko attribuuttisolmuja USES- ja PRODUCES-suhteilla. Lisäksi prosessilla voi olla siitä vastaava toimija, joka liitetään prosessiin IS_RESPONSIBLE_FOR-suhteella. *Toimija* onkin ainoastaan prosesseihin liittyvä solmutyyppi, joka kertoo prosessista vastaavan tahon.

Tuote vastaa jäljitettävyysgraafin yhteydessä määriteltyä tuotetta. Tuotteet liittyvät tietokannassa objekteihin ja prosesseihin. Tuotteen ja objektin välisellä IS_A-suhteella ilmaistaan, minkä tuotteen ilmentymä kukin objekti on. Tuotteen ja prosessin välinen PRODUCT_PORTION-suhde puolestaan kertoo, miten esimerkiksi kussakin prosessissa tuotettavat päästöt jakautuvat prosessissa käsiteltävien tuotteiden kesken.

Objektit ovat tuotteiden esiintymiä. Objekti voi jäljitystarkkuudesta riippuen olla joko yksittäinen objekti, useamman objektin muodostama kokonaisuus tai fyysisesti tunnistettavissa olevaa materiaalimassaa. Objektit liittyvät tuotteisiin, prosesseihin ja toisiin objekteihin. Tuotteeseen liittyvällä IS_A-suhteella ilmaistaan, mitä tuotetta objekti edustaa. Objektin ja prosessin välinen ROUTE-suhde kertoo, mihin prosesseihin objekti on osallistunut. IS_MADE_OF-suhde yhdistää objektin niihin objekteihin, joita on joko käytetty objektin valmistamisessa (solmuun tulevat suhteet) tai joiden valmistamiseen objektia on käytetty (solmusta lähtevät suhteet).

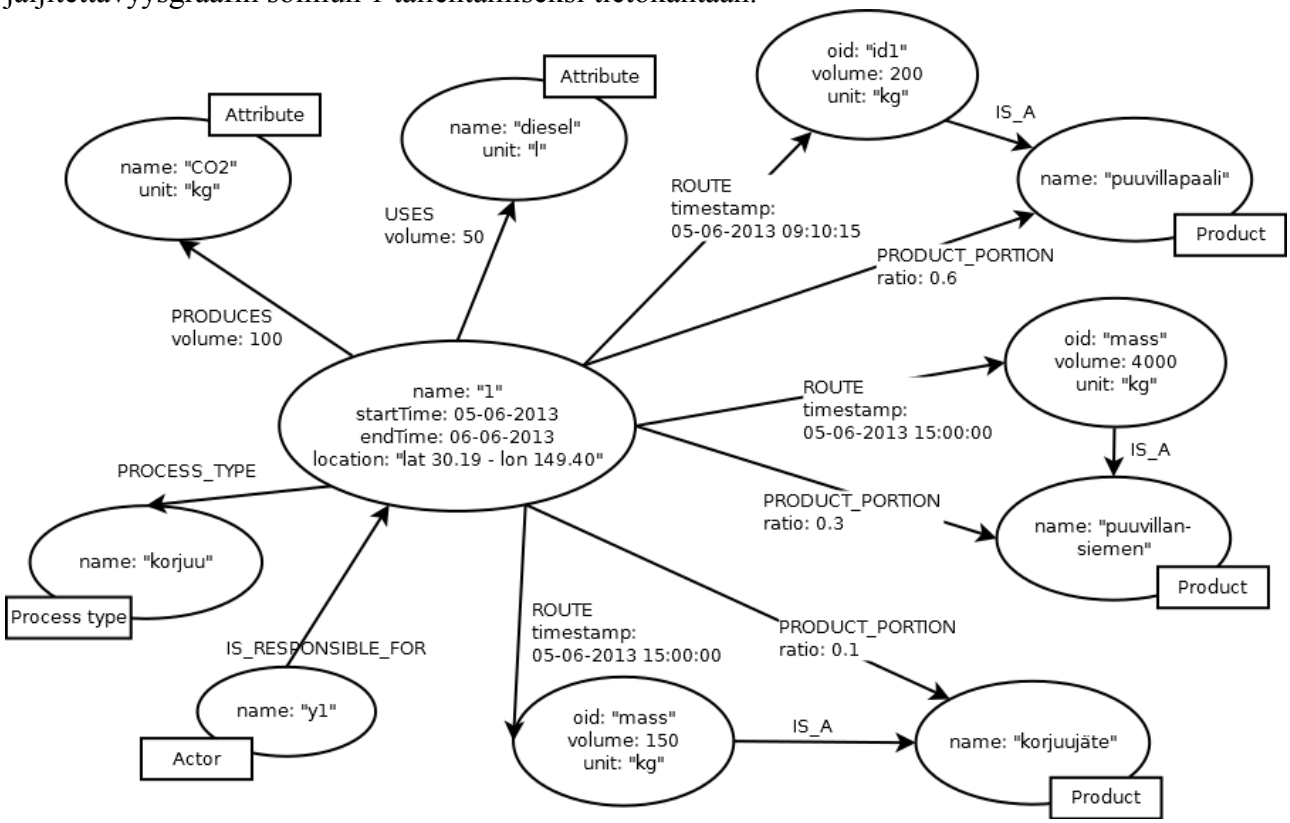
Objektilla voi olla IS_MADE_OF-suhde myös vain yhteen objektiin. IS_MADE_OF-suhteeseen liittyvä ”transformationFunction”-ominaisuus on kerroin, jota käytetään attribuuttien kumuloituneiden arvojen laskemisessa.

Attribuuteilla tarkoitetaan niitä prosessien kuluttamia ja tuottamia resursseja ja päästöjä, jotka eivät ole tuotteita. Attribuutit liittyvät tietokannassa prosesseihin, joissa niitä kulutetaan tai josta ne aiheutuvat. Prosessi yhdistetään attribuuttiin joko USES- tai PRODUCES-suhteella sen mukaan, onko kyseessä syöteattribuutti (prosessissa kuluva attribuutti) vai tulosattribuutti (prosessin tuottama attribuutti). Attribuuttien kumuloituneita arvoja ei tallenneta tietokantaan erikseen, koska ne voidaan tarvittaessa laskea prosessikohtaisten arvojen perusteella.

Dokumentit ovat erilaisia tekstidokumentteja, jotka voivat liittyä mihin tahansa solmuun. Jäljitettävyysgraafissa ne voidaan kuvata informaatioattribuutteina, mutta tietokannassa jokaista dokumenttia varten luodaan oma tietokantasolmu. Dokumenteista tallennetaan tietokantaan vain tunnus, jonka perusteella ne voidaan etsiä käänteishakemistosta. Dokumenttisolmujen yhteydet muihin solmuihin on merkitty harmailla nuolilla, jotta kuva olisi helppolukuisempi. Dokumentit yhdistetään solmuihin IS_ABOUT-tyyppisellä suhteella.

5.3. Jäljitettävyysgraafi tietokannassa

Kuvassa 9 on esimerkki edellisessä luvussa esitellyn jäljitettävyysgraafin tallentamisesta tietokantaan. Kuvaan on merkitty kaikki solmut ja suhteet, jotka tarvitaan kohdassa 4.4.3 annetun jäljitettävyysgraafin solmun 1 tallentamiseksi tietokantaan.

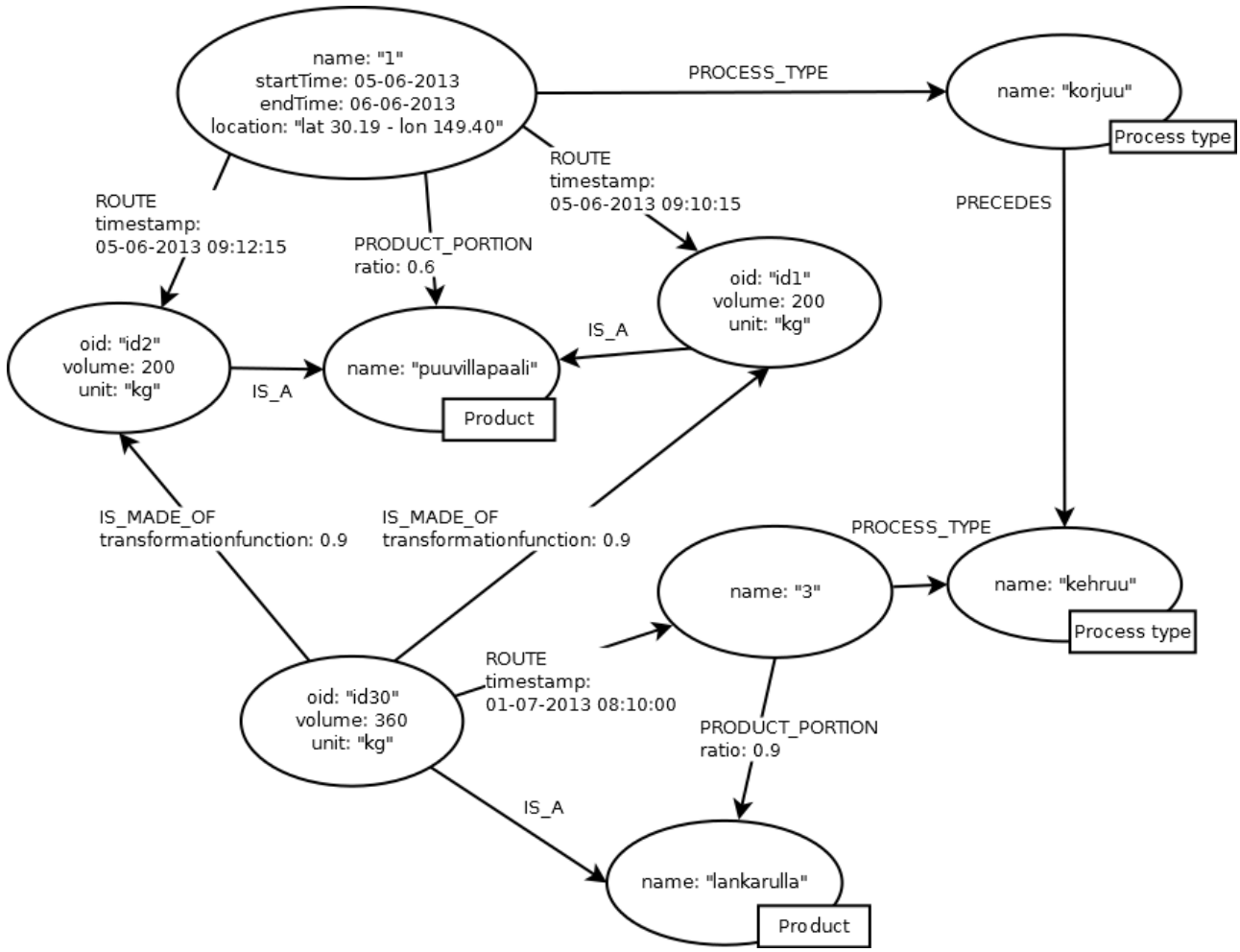


Kuva 9. Jäljitettävyysgraafin solmu Neo4j-tietokannassa.

Prosessisolmu esitetään tietokannassa solmuna. Sen tunniste ja informaatioattribuutti ”sijainti” ovat solmun ominaisuuksia. Tuote-erä (esimerkiksi ⟨Puuvillapaali, 2 000 kg, {id1, …, id10}, 0,6⟩) esitetään tuotesolmun (puuvillapaali), objektisolmun (id1) ja niiden välisen PRODUCT_PORTION-suhteen avulla. Suhteen ominaisuudeksi lisätään tuote-erän suhdeluku, eli tässä tapauksessa 0,6. Kuvassa 9 näkyy vain yksi objektia edustava tietokantasolmu, mutta vastaavasti prosessiin liitetään muutkin prosessiin liittyvät objektit id2:sta id10:een. Jokainen objekti myös liitetään IS_A-suhteella vastaavaan tuotesolmuun. Tuote-erän koko (paino) voidaan laskea kaikkien siihen kuuluvien objektien yhteiskoosta (tässä oletetaan, että jos objektit ovat samaa tuotetta, niiden koko ilmoitetaan samassa mittayksikössä).

Prosessiin liittyvät tulos- ja syöteattribuutit, esimerkiksi ⟨CO_2, tulos, 100 kg, 100 kg⟩, tallennetaan tietokantaan luomalla prosessisolmun ja attribuuttisolmun välille suhde, jonka ominaisuudeksi asetetaan tieto siitä, paljonko kyseistä attribuuttia syntyy tai kuluu prosessissa.

Kuvassa 10 on esimerkki jäljitettävyysgraafin kaaren tallentamisesta tietokantaan. Kuten kuvasta näkyy, prosessisolmujen 1 ja 3 välinen yhteys muodostuu tietokannassa prosesseihin osallistuvien objektien kautta. Prosessisolmujen välille ei siis lisätä tietokannassa erikseen suhdetta.



Kuva 10. Jäljitettävyysgraafin kaari Neo4j-tietokannassa.

Objektikuvaus ⟨{id1, id2}, id30⟩ esitetään tietokannassa lisäämällä objektista id30 IS_MADE_OF-suhde molempiin objekteihin, joista se koostuu (id1 ja id2). Näin objektien väliset suhteet voidaan tallentaa luontevasti suorana viittauksena objektista toiseen. Koska tietokannan suhteet ovat aina suunnattuja, IS_MADE_OF-suhteen suunnasta voidaan päätellä, mistä objekteista kukin objekti koostuu.

IS_MADE_OF-suhteelle lisätään ”transformationFunction”-ominaisuus, jonka arvoa käytetään kumuloituneiden attribuuttien arvojen laskemiseen. Se saadaan kertomalla objektiin liittyvän tuote-erän suhdeluku objektin osuudella siirtyvästä tuote-erästä. Esimerkkitapauksessa puuvillapaaliobjektin (id1) ja lankarullaobjektin (id30) välisen suhteen ”transformationFunction”-arvo on 0,9, koska lankarullan osuus tuote-erästä on 90 % ja puuvillapaalit käytetään lankarullaan kokonaan (0,9 · 1).

Kaaren johdettujen attribuuttien arvoja ei tallenneta erikseen, koska ne voidaan laskea muiden tietokantaan tallennettujen arvojen perusteella. Esimerkiksi solmujen 1 ja 3 välisen johdetun attribuutin arvo voidaan laskea kulkemalla prosessisolmusta 3 ROUTE-suhdetta pitkin objektisolmuun (tässä tapauksessa id30), josta on IS_MADE_OF- ja IS_A-suhteiden välityksellä reitti ”puuvillapaali”-tuotesolmuun ja siitä prosessisolmuun. Tuotesolmun ja prosessisolmun välisen PRODUCT_PORTION-suhteen ominaisuudesta saadaan suhdeluku 0,6, joka kerrotaan edellä kuvatulla reitillä olevien puuvillapaaliobjektien osuudella prosessin kaikista puuvillapaaliobjekteista. Kaikki objektit eivät näy kuvassa, mutta reittiin sopivia objekteja olisi viisi ja objekteja yhteensä kymmenen, joten kerroin on 0,5. Näin tulokseksi saadaan 0,6 · 0,5 = 0,3. Kun kerrotaan prosessiin 1 liittyvien attribuuttien arvot 0,3:lla, saadaan tulokseksi jäljitettävyysgraafin kaaren johdettujen attribuuttien arvot (esimerkiksi dieselin arvoksi 50 litraa · 0,3 = 15 litraa).

Edellä kuvattu attribuuttien arvojen laskeminen graafin suhteita pitkin kulkemalla vaikuttaa monimutkaiselta, mutta useista siirtymistä huolimatta tämän kaltaiset kyselyt ovat graafitietokannoissa tehokkaita, koska tietokannat on suunniteltu nimenomaan niitä varten [Robinson *et al.*, 2013].

6. Tiedonhaku tietokannasta

Edellisessä luvussa esiteltiin, miten jäljitettävyysgraafi tallennetaan graafitietokantaan. Tässä luvussa kerrotaan, mitä tietoja tietokannasta ja siihen yhdistetystä käänteishakemistosta voidaan hakea ja millaisia tietoja hakutuloksista esitetään. Hakemisella tarkoitetaan tässä kyselyitä, jotka tietokantaan ja käänteishakemistoon voidaan tehdä tässä työssä toteutetulla hakuohjelmalla.

Haun tavoitteena on selvittää, miten jokin termi liittyy tuotteeseen tai prosessiin. Hakuohjelmassa haut tehdään syöttämällä hakuavain graafisen käyttöliittymän kautta. Myös haun rajaukset valitaan käyttöliittymässä. Käyttöliittymä ja varsinaiset tietokantakyselyt esitellään tarkemmin seuraavassa luvussa, ja tässä luvussa esitellään yleisesti, mitä tietoja ohjelmalla voidaan hakea.

6.1. Haun kohteet ja rajaukset

Haun kohteena voi olla prosessityyppi, prosessi, tuote, objekti, attribuutti, toimija tai dokumentti. Haun voi kohdistaa joko yhteen tai kuinka moneen tahansa kohdetyyppiin. Haku on jaettu sanahakuun ja numerohakuun. Sanahaku tarkoittaa tekstihakua, jossa kohteita haetaan hakusanoilla. Numerohaussa kohteita haetaan päivämäärän perusteella.

Hakua voi rajata sen mukaan, millaisia ominaisuuksia kohteista on tallennettu tietokantaan. Poikkeuksena tästä ovat dokumentit, joita voi lisäksi hakea otsikon ja sisällön perusteella: tällöin haku kohdistuu tietokannan sijaan käänteishakemistoon. Taulukossa 1 luetellaan mahdolliset hakukohteet ja ominaisuudet, joiden perusteella hakua voidaan rajata hakuohjelmassa.

Haun kohde	**Haun rajaukset sanahaussa**	**Haun rajaukset numerohaussa**
Prosessityyppi	Nimi	–
Prosessi	Tunnus, sijainti	Alkuaika, loppuaika
Tuote	Nimi	–
Objekti	Tunnus	Aikaleima
Attribuutti	Nimi	–
Toimija	Nimi	–
Dokumentti	Tunnus, otsikko, sisältö	–

Taulukko 1. Haun kohteet ja rajaukset.

Sanahaussa kohteita voi hakea niiden ominaisuuksien perusteella, joista on luotu tietokannan sisäinen käänteishakemisto. Kuten taulukosta 1 näkyy, nämä ominaisuudet ovat enimmäkseen kohteita yksilöiviä tietoja, kuten nimi tai tunnus. Esimerkiksi objektin kokoa tai yksikköä ei ole lisätty käänteishakemistoon, koska ne eivät oletettavasti ole kiinnostavia tietoja haun kannalta.

Objekteja ja prosesseja voi sanahaun lisäksi hakea niihin liittyvien aikaleimojen perusteella. Tätä kutsutaan numerohauksi. Numerohaulla voidaan hakea objektit, jotka on valmistettu ennen tiettyä päivämäärää tai sen jälkeen tai jotka on valmistettu tietyllä aikavälillä. Samoin voidaan

hakea prosesseja, tosin prosessien haussa rajaus voi kohdistua prosessin alku- tai loppuaikaan tai molempiin.

6.2. Hakutulokset

Haun tuloksena voi olla joukko dokumentteja, prosesseja, prosessityyppejä, tuotteita, objekteja, toimijoita tai attribuutteja sen mukaan, mitä kohdetyyppejä on valittu haettavaksi. Sama kysely voi tuottaa tuloksia myös useammasta kuin yhdestä edellä luetellusta tyypistä. Esimerkiksi hakuavaimella ”diesel” voidaan saada tulokseksi sekä attribuutti että dokumentteja, joissa mainitaan ”diesel”.

Haun toteutusta on rajattu ohjelmassa niin, että haun tuloksena saadaan vain kohteita, joiden ominaisuuksissa annettu hakuavain esiintyy (tai dokumenttien tapauksessa myös otsikossa tai sisällössä). Esimerkiksi edellä mainittu ”diesel”-haku ei palauta suoraan hakutuloksena prosesseja, joissa dieseliä kulutetaan, vaikka ne dieseliin sinänsä liittyvätkin. Tämä tieto saadaan avaamalla diesel-nimisen attribuutin tiedot, joissa luetellaan, mitkä prosessit dieseliä kuluttavat.

Hakutulokset näytetään tyypeittäin luetteloissa. Hakutuloksesta voidaan valita kohde, jonka tiedot näytetään erillisessä ikkunassa. Taulukossa 2 luetellaan, mitä tietoja kustakin kohdetyypistä kerrotaan, kun kohteen tiedot avataan. Kustakin kohdetyypistä näytetään ohjelmassa kohteesta tallennetut ominaisuudet, kuten nimi tai koko. Lisäksi kerrotaan, millaisia olioita kohteeseen liittyy: esimerkiksi prosessista kerrotaan siinä käsitellyt objektit.

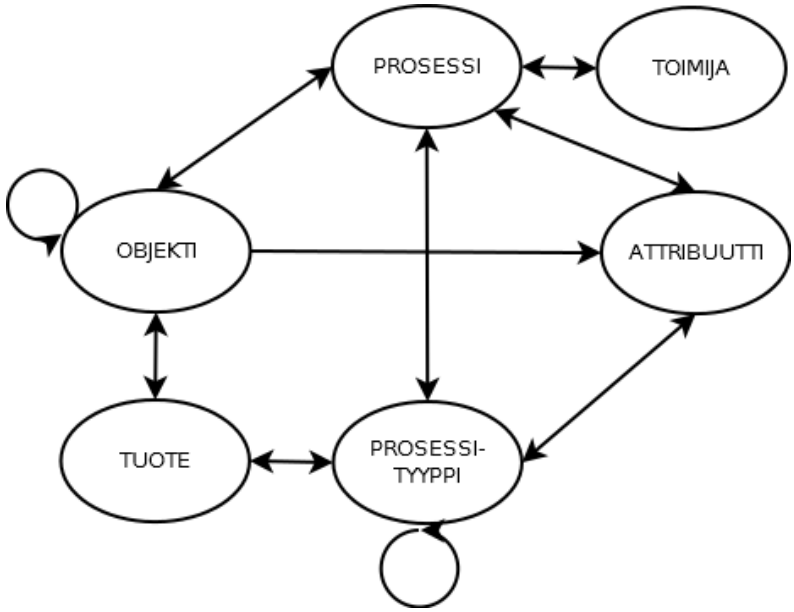
Hakutulos	**Mitä tietoja kohteesta saadaan**
Prosessityyppi	Nimi, lopputuote, esiintymät (prosessit), toimitusketjun edeltävä ja seuraava prosessityyppi, dokumentit
Prosessi	Tunnus, prosessityyppi, prosessista vastaava toimija, prosessiin liittyvät objektit ja attribuutit, prosessin alku- ja loppuaika, sijainti, dokumentit
Tuote	Nimi, esiintymät (objektit), prosessityyppi, jonka lopputuote tuote on, dokumentit
Objekti	Tunnus, objektin edustama tuote, koko (esimerkiksi paino tai tilavuus), prosessi, aikaleima, ympäristökuormitus, historia, dokumentit
Attribuutti	Nimi, yksikkö, prosessit, joissa attribuuttia kuluu tai tuotetaan, dokumentit
Toimija	Nimi, ominaisuudet, dokumentit
Dokumentti	Tunnus, otsikko, oliot, johon dokumentti liittyy

Taulukko 2. Haun tuloksena saatavat tiedot.

Jos taulukossa 2 lueteltuja tietoja verrataan tietokantakaavioon, huomataan, että jokaiseen kohteeseen liittyen luetellaan käytännössä ne oliot, jotka ovat kohteen naapurisolmuja tietokannassa. Tässä työssä toteutettua hakua ja hakutulosten esitystapaa voikin ajatella eräänlaisena graafissa kulkemisena. Sana- tai numerohaulla etsitään lähtösolmu, jonka kautta graafin (eli

tietokannan) sisältöä voidaan selata siirtymällä eri kohteiden tietojen välillä. Hakuohjelman pääpaino on näin olioiden välisten suhteiden tutkimisessa.

Kuvassa 11 esitetään, millaisia siirtymiä hakuohjelma mahdollistaa eri kohteiden välillä. Kuvasta on selkeyden vuoksi jätetty pois dokumentit. Koska dokumentit voivat liittyä kaikkiin muihin kohdetyyppeihin, dokumenttien tiedoista voidaan siirtyä jokaiseen muuhun kohdetyyppiin ja päinvastoin.



Kuva 11. Siirtymät kohteiden välillä.

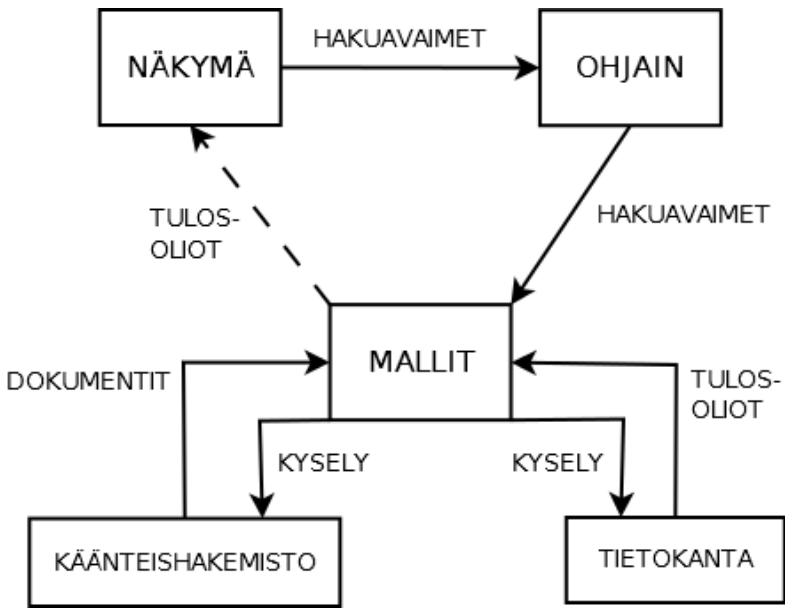
Kuvassa 11 olevat nuolet kuvaavat, miten kohteiden välillä voidaan siirtyä. Esimerkiksi objektin tiedoista voidaan siirtyä tarkastelemaan siihen liittyvään tuotteen tietoja, objektin valmistusprosessin tietoja tai objektien ympäristökuormitukseen liittyvien attribuuttien tietoja. Objektista on nuoli myös itseensä, mikä tarkoittaa, että objektista voidaan siirtyä myös objekteihin, joita on käytetty objektin valmistamisessa tai joiden valmistamiseen objektia on käytetty.

7. Tekninen toteutus

Jäljitettävyysgraafin tietokantatoteutuksen ympärille on toteutettu Java-ohjelmointikielellä hakuohjelma, jolla käyttäjä voi tehdä avainsanahakuja ja selvittää, miten haettu termi liittyy tuotteeseen tai prosessiin. Hakuohjelmassa on graafinen käyttöliittymä, johon hakulause syötetään ja jossa hakutulokset näytetään. Tiedonhaun kokeilua varten tietokantaan on syötetty kuvitteellista dataa aiemmin kuvatusta T-paidan valmistusprosessista. Ohjelma on toteutettu Netbeans IDE -ohjelmointiympäristössä.

7.1. Arkkitehtuuri

Ohjelma on toteutettu MVC (Model-View-Controller) -arkkitehtuurin mukaisesti. Sen pääkomponentit ovat näkymät, ohjain ja mallit sekä tietokanta ja käänteishakemisto. Pääkomponenttien keskinäiset yhteydet ja tiedonkulku on esitetty kuvassa 12. Komponentit on merkitty kuvaan suorakulmioilla ja komponenttien välillä kulkeva tieto nuolilla, joiden suunta osoittaa, mihin suuntaan tieto kulkee.



Kuva 12. Ohjelman pääkomponentit.

Ohjelman päänäkymää käytetään sekä hakuavainten syöttämiseen että hakutulosten esittämiseen. Sen yläosassa ovat hakuun liittyvät valinnat, ja alaosassa on tulosnäkymä, joka on jaettu välilehtiin. Kun tulosnäkymästä valitaan hiirellä jokin kohde, avautuu uusi ikkuna, jossa valitun kohteen tiedot näytetään.

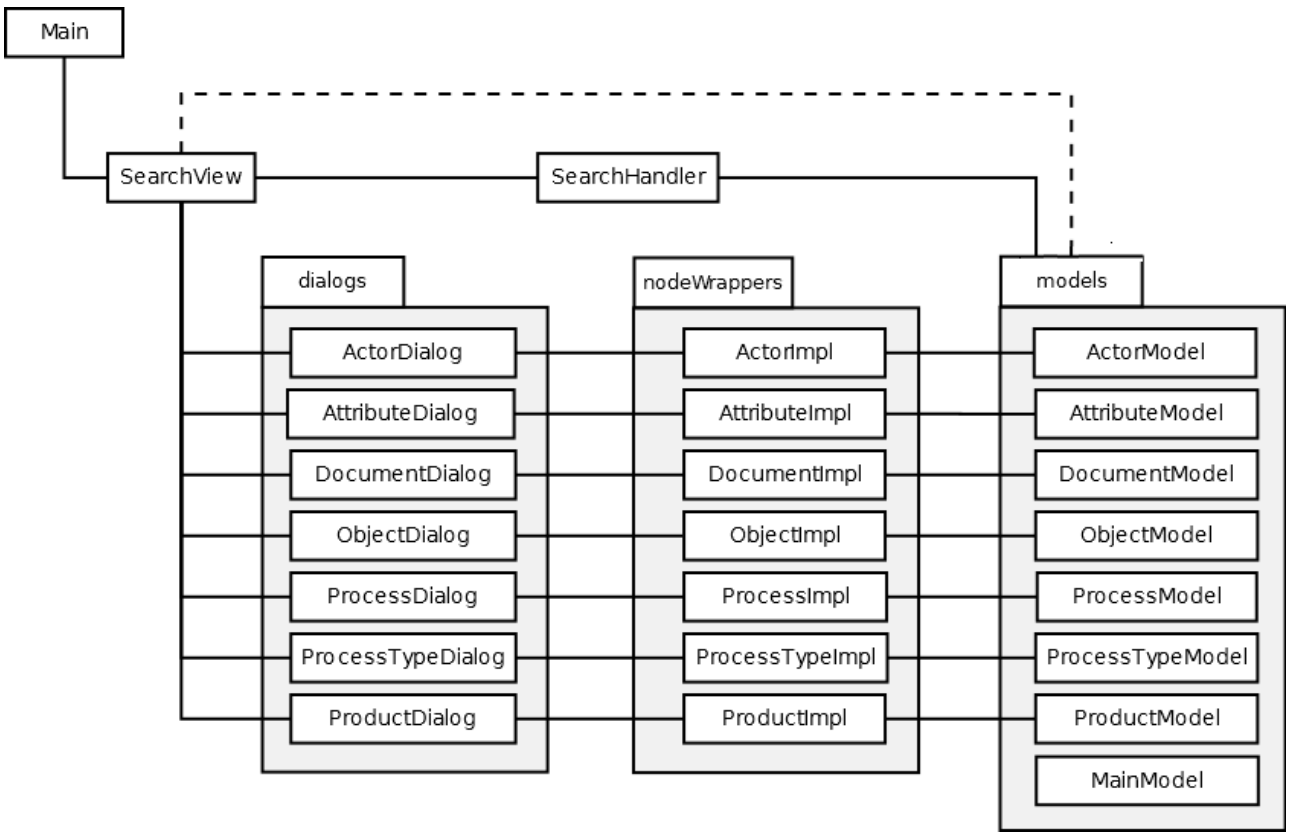
Ohjain (SearchHandler-luokka) vastaanottaa käyttäjän syötteen ja pyytää malleja hakemaan syötettä vastaavat tiedot. Tässä tapauksessa se välittää malleille käyttäjän syöttämät hakuavaimet ja tiedon siitä, mihin ominaisuuksiin haku kohdistuu. Ohjain myös tarkastaa syötetyn merkkijonon ja käsittelee sitä niin, että mallit voivat hyödyntää sitä. Se esimerkiksi poistaa tyhjät merkit

merkkijonon alusta ja lopusta ja tarvittaessa muuntaa päivämäärämuotoisen hakuavaimen tietojen tallennuksessa käytettyyn muotoon (Javan Long-luokan olioksi).

Mallit ovat yhteydessä tietokantaan ja käänteishakemistoon. Ne muun muassa tutkivat, millaisia ominaisuuksia tietokannan solmuilla on, ja palauttavat ominaisuusluettelot näkymän käytettäväksi. Näin hakunäkymässä valittavissa olevat ominaisuudet ovat samat kuin tietokannassa olevat solmujen ominaisuudet. Mallit myös suorittavat kaikki tietokantaan kohdistuvat kyselyt, niin käyttäjän tekemät haut kuin ohjelman vaatimat sisäiset haut. Jokaiselle ohjelmassa käsiteltävälle oliotyypille on oma mallinsa, johon on koottu kyseiseen oliotyyppiin liittyvät tietokantakyselyt. Oliotyypit ovat prosessi, prosessityyppi, objekti, tuote, attribuutti, toimija ja dokumentti.

Käänteishakemisto nopeuttaa tiedonhakua dokumentteihin kohdistuvissa hauissa. Käänteishakemistoa käsitellään dokumentteihin liittyvistä hauista vastaavan mallin kautta. Käänteishakemistosta kerrotaan tarkemmin seuraavassa alakohdassa.

Kuvassa 13 on ohjelman luokkakaavio, joka kuvaa ohjelman rakenteen hieman tarkemmin. Luokkakaaviosta on selkeyden vuoksi jätetty pois luokat, joilla ei ole merkitystä ohjelman toiminnan kuvauksen tai ymmärtämisen kannalta. Tällaisia luokkia ovat jotkin rajapinnat, enumeraatiot ja Javan valmiista luokista periytetyt apuluokat, joita käytetään apuna näkymien luomisessa.



Kuva 13. Ohjelman Java-luokat.

Kuvaan on merkitty harmaalla kolme luokkapakettia, joiden sisällä olevat luokat ovat toiminnaltaan samankaltaisia. Dialogs-paketti sisältää olioiden tietojen näyttämisessä käytettävät

näkymät. NodeWrappers-pakettiin kuuluvat luokat ovat apuluokkia, joiden sisään Neo4j-tietokannan solmut tallennetaan. Models-paketti sisältää mallit, joista kukin vastaa tietyntyyppisiin olioihin liittyvistä tietokantatoiminnoista.

Kuvassa 13 oleva katkoviiva, joka on merkitty SearchView-luokan ja mallien väliin, tarkoittaa, että luokkien välillä on epäsuora yhteys tarkkailijamallin (observer pattern) kautta. Tarkkailijamalli on MVC-mallin kanssa usein käytettävä suunnittelumalli [OODesign, 2015]. Siinä yksi olio on tarkkailtava, johon liitetään muita olioita (tarkkailijoita), joiden halutaan reagoivan, kun tarkkailtavan tila muuttuu. Tilan muutoksesta ilmoitetaan automaattisesti tarkkailijoille, jotka ilmoituksen saadessaan toimivat halutulla tavalla. Esimerkiksi tässä sovelluksessa tarkkailijamallia käytetään hakutulosten näyttämiseen. SearchView-luokan näkymäolio on tarkkailija, jolle mallit, eli tarkkailtavat, ilmoittavat hakutulosten löytymisestä, kun käyttäjä tekee haun.

7.2. Käänteishakemisto

Neo4j-tietokannan lisäksi hakuohjelmaan kuuluu Apache Lucene -hakukirjastolla toteutettu käänteishakemisto. Apache Lucene on avoimeen lähdekoodiin perustuva, kokonaan Javalla toteutettu tiedonhakukirjasto [Apache Lucene Core, 2015], joka soveltuu tekstipohjaisen tiedon indeksointiin ja hakuun. Hakuohjelman käänteishakemistoon on indeksoitu joukko tekstidokumentteja, kuten lähetyslistoja, käyttöturvallisuustiedotteita ja raportteja, jotka voisivat sisältönsä puolesta liittyä kohdassa 4.3 kuvattuun esimerkkisovellukseen. Tämä hakuohjelmaa varten luotu käänteishakemisto on eri kuin alikohdassa 3.2.2 mainittu Neo4j-tietokannan sisäinen indeksi, jota käytetään vain kyseisen tietokannan operaatioihin.

Lucene-kirjastossa indeksoinnin ja haun keskeinen käsite on dokumentti (Lucenen Document-Java-luokka). Dokumentilla tarkoitetaan Lucenessa joukkoa kenttiä, joilla on nimi ja tekstimuotoinen sisältö. [Apache Lucene API, 2015a] Esimerkiksi tässä sovelluksessa edellä mainitut tekstidokumentit muodostetaan kentistä ”polku” (tiedoston sijainti), ”otsikko”, ”sisältö” ja ”tunnus”. Koska käänteishakemistoon lisättävät dokumentit koostuvat vapaasti määritettävistä kentistä, indeksoitavan aineiston ei tarvitse olla varsinaisesti dokumentin muodossa, vaan mitä tahansa tekstimuotoista sisältöä voidaan indeksoida. Tämä mahdollistaa esimerkiksi Neo4j-tietokannassa käytettävän sisäisen käänteishakemiston, jossa yksittäinen solmu voidaan esittää dokumenttina ja solmun ominaisuudet dokumentin kenttinä.

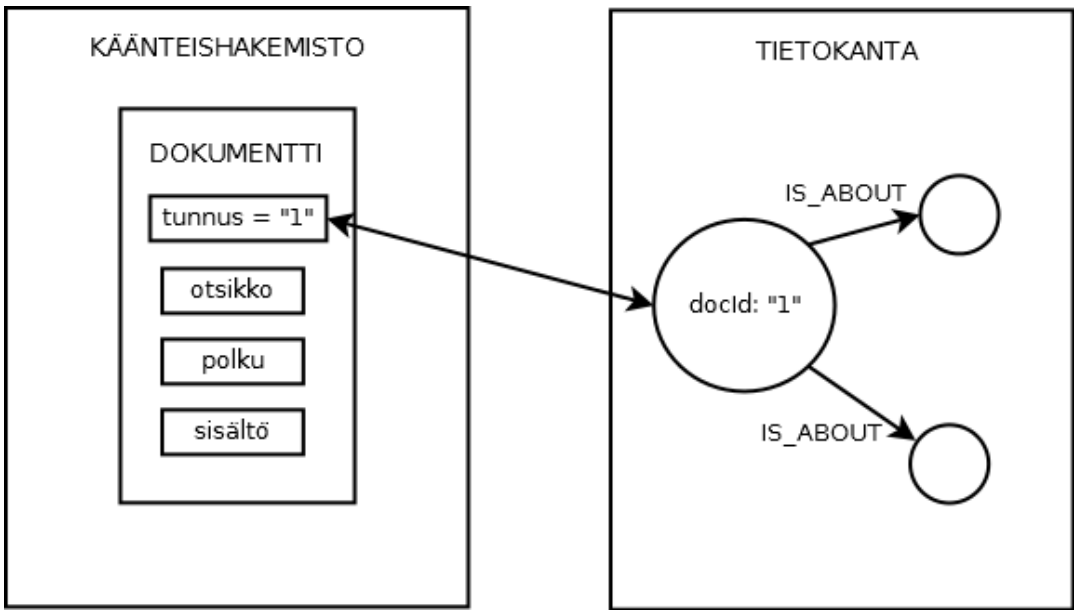
Kun käänteishakemisto luodaan, dokumentti (Document-luokan olio) lisätään Lucenen IndexWriter-luokan olioon, joka luo käänteishakemiston ja ylläpitää sitä. Kun käänteishakemistosta haetaan dokumentteja, merkkijonomuotoisesta hakuavaimesta luodaan Lucenen omalla jäsentimellä Query-olio, joka välitetään kyselyn suorituksesta vastaavan IndexSearcher-olion hakumetodille. Haun tuloksena on joukko dokumentteja, jotka täsmäävät kyselyyn.

Kyselyitä voi muokata ja täsmentää useilla tavoilla. Boolen operaattoreista käytössä ovat AND, OR, NOT, ”+” ja ”–”. Jos käyttäjä on syöttänyt monta hakuavainta, ne yhdistetään oletusarvoisesti OR-operaattorilla, jos mitään muuta operaattoria ei ole käytetty. Kyselyssä voidaan määrittää myös kentät, joihin haku kohdistetaan. Esimerkiksi kysely ”*otsikko:diesel*” hakee dokumentit, joiden

otsikkokentässä mainitaan diesel. Haussa voi käyttää vain kenttiä, jotka on määritetty indeksointivaiheessa. [Apache Lucene API, 2015b]

Käänteishakemisto ja tietokanta liittyvät toisiinsa ohjelman DocumentModel-luokan operaatioiden välityksellä. DocumentModel-luokka vastaa kaikista dokumentteihin liittyvistä kyselyistä. Tieto dokumenttien ja muiden ohjelmassa käsiteltävien olioiden yhteydestä on tallennettu vain graafitietokantaan. Dokumenttia kuvaavalle tietokantaoliolle luodaan IS_ABOUT-suhde kaikkiin solmuihin, joiden kuvaamiin olioihin dokumentti liittyy. Dokumenteista tallennetaan tietokantaan vain tunnus. Dokumentin tunnus indeksoidaan tietokannan sisäiseen käänteishakemistoon, jotta yksittäinen dokumentti voidaan hakea nopeasti tietokannasta.

Dokumenttihaku kohdistuu aina käänteishakemistoon, ja tietokannasta dokumentteja haetaan tekstihaulla vain silloin, kun on tarpeen löytää tietty solmu esimerkiksi graafissa kulkemisen lähtösolmuksi. Käänteishakemiston ja tietokannan välinen yhteys dokumentin tunnuksen välityksellä on esitetty kuvassa 14. Dokumenttia kuvaavalla Lucenen dokumenttioliolla on siis sama tunnus kuin tietokantaan tallennettavalla dokumenttisolmulla, ja dokumentti voidaan hakea kummasta tahansa tämän tunnuksen perusteella.



Kuva 14. Dokumenttien tallennus käänteishakemistoon ja tietokantaan.

7.3. Tietokanta

Graafitietokanta on hakuohjelmaa varten luotu paikallisesti kiintolevylle, ja sitä käytetään ohjelmassa Neo4j:n Java-ohjelmointirajapinnan kautta. Näin tietokannan solmuja ja suhteita voidaan käsitellä ohjelmassa olioina samalla tavalla kuin muita olioita. Tietokannan operaatioiden

käyttöä varten ohjelmassa luodaan GraphDatabaseService-luokan[2] olio, jonka kautta tietokantaa päästään käyttämään (koodiesimerkki 1).

```
GraphDatabaseService graphDb =
        graphDbFactory.newEmbeddedDatabase(TIETOKANTAPOLKU);
```

Koodiesimerkki 1. Tietokantaolion luominen.

GraphDatabaseService-luokan operaatioita käytetään muun muassa tietokannan solmujen luomiseen ja solmujen etsimiseen. Solmuja voidaan etsiä esimerkiksi tunnisteiden (tietokannassa käytettävät sisäiset tunnisteet) tai tunnusten (Label-luokka) avulla. Koodiesimerkissä 2 tietokantaan luodaan ensin solmu, jolle lisätään tunnus ”PROCESS_TYPE”. Sen jälkeen tietokannasta etsitään tunnuksen perusteella kaikki prosessityypit, eli solmut, joille on lisätty tunnus ”PROCESS_TYPE”. Tuloksena on iteraattori, joka sisältää halutun tyyppiset solmut.

```
Node processType = graphDb.createNode(Labels.PROCESS_TYPE);

ResourceIterator<Node> processTypes = graphDb.findNodes(Labels.PROCESS_TYPE);
```

Koodiesimerkki 2. Kaikkien prosessityyppisolmujen hakeminen tietokannasta.

GraphDatabaseService-luokkaa käytetään myös graafissa kulkemiseen tarvittavien TraversalDescription-kuvausten luomiseen sekä tietokantakaavion ja indeksien hallintaan. Graafissa kulkemista ja indeksien käyttöä havainnollistetaan seuraavan kohdan esimerkkikyselyjen yhteydessä.

Tietokannassa käytetään Neo4j:n vanhempaa indeksointimenetelmää, jossa solmut ja suhteet on lisättävä indeksiin manuaalisesti. Uudemmassa indeksoinnissa indeksit luodaan tunnusten perusteella, ja solmut lisätään indeksiin automaattisesti, kun niille lisätään tunnus, johon liittyy indeksi. Tähän ohjelmaan on kuitenkin valittu vanhempi indeksointimenetelmä, koska se tukee monipuolisemmin esimerkiksi numeroiden indeksointia ja solmujen hakemista numeroarvojen perusteella [Neo4j Manual, 2015d]. Ohjelmassa tarvitaan numeroarvojen käsittelyä objektien ja prosessien päivämäärähakuun, koska päivämäärät on tallennettu tietokantaan numeromuodossa.

Kun tietokannasta palautetaan sovellukseen solmuja esimerkiksi edellä kuvatun findNodes()-metodin tuloksena, ne palautetaan Node-tyyppisinä olioina. Jotta erityyppisten solmujen käsittely ohjelmassa helpottuisi, näistä olioista luodaan malleissa olion tyyppiä vastaavan kääreluokan olio. Esimerkiksi objektisolmuista luodaan ObjectImpl-tyyppisiä olioita. Mallien ulkopuolella käsitellään vain kääreluokkien olioita. Kääreluokat on lueteltu kuvan 13 nodeWrappers-paketissa.

Myös suhteita käsitellään tietokannassa olioina (Relationship-luokka), eikä esimerkiksi solmujen välisillä viittauksilla. Kun suhde luodaan, sille on määritettävä suhdetyyppi. Koska ohjelman suhdetyypit ovat ennalta tiedossa eivätkä ne muutu ohjelman ajon aikana, suhdetyyppejä varten on luotu enumeraatio. Enumeraatio on luokka, jossa luetellaan kaikki mahdolliset luokan

[2] ObjectImpl-luokkaa lukuun ottamatta kaikki luvussa 7.3 mainittavat luokat ovat Neo4j:n ohjelmointirajapintaan kuuluvia luokkia.

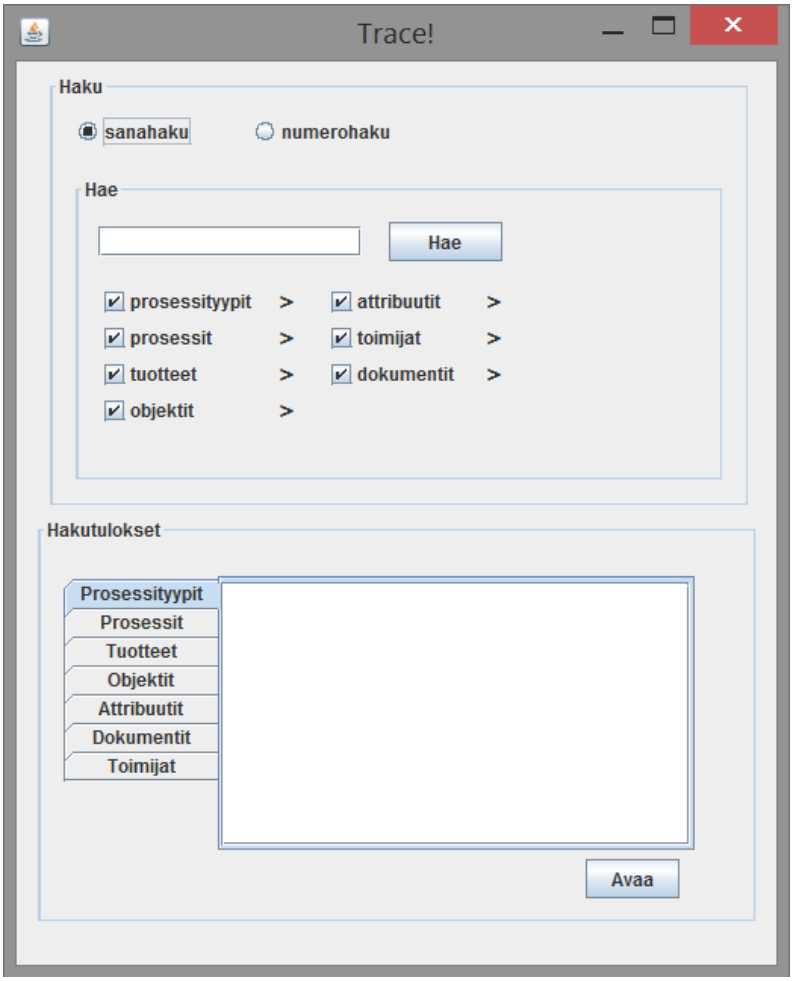
ilmentymät, eli arvot. Sen avulla suhdetyyppeihin voidaan viitata ohjelmassa yhtenäisesti enumeraation arvoilla. Ohjelmassa käytettävät suhdetyypit luetellaan koodiesimerkissä 3.

```
public enum RelationshipTypes implements RelationshipType {
      IS_A, IS_ABOUT, IS_MADE_OF, IS_RESPONSIBLE_FOR, PRODUCES, ROUTE,
      PRODUCT_PORTION, PROCESS_TYPE, PRECEDES, USES
}
```

Koodiesimerkki 3. Tietokannan suhdetyypit.

7.4. Kyselyt

Kyselyt tehdään hakuohjelman graafisen käyttöliittymän kautta syöttämällä hakuavain pääikkunan tekstikenttään. Kuvassa 15 on kuvankaappaus käyttöliittymän päänäkymästä.

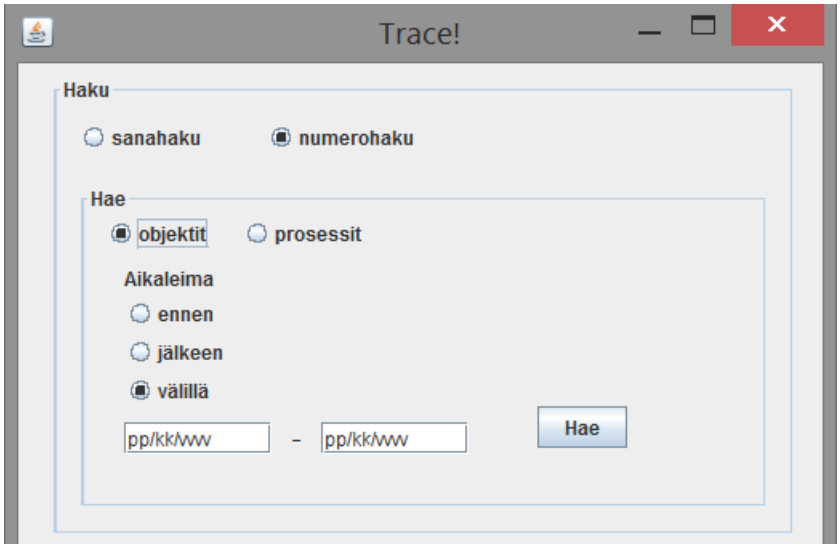


Kuva 15. Kuvankaappaus ohjelman käyttöliittymästä.

Haun kohteet ja hakutulokset on esitelty tarkemmin edellisessä luvussa. Avainsanahaku kohdistuu sekä varsinaiseen tietokantaan että dokumenteista muodostettuun käänteishakemistoon. Sanahaku kohdistuu tietokantasolmujen ominaisuuksiin. Haun tuloksena voi olla joukko dokumentteja, prosesseja, prosessityyppejä, tuotteita, objekteja, toimijoita tai attribuutteja. Haku voi tuottaa tuloksia myös useammasta kuin yhdestä edellä luetellusta oliotyypistä. Oletuksena ohjelma

hakee annettuja hakuavaimia valittujen hakukohteiden kaikista ominaisuuksista, mutta haun voi halutessaan kohdistaa vain tiettyihin ominaisuuksiin. Koska tietokannan sisäinen indeksointi perustuu Lucene-hakukirjastoon, kyselyn laatimisessa voidaan käyttää Lucenen kyselyrakenteita, joista annettiin esimerkkejä kohdassa 7.2.

Toinen tapa hakea tietoa järjestelmästä on päivämäärähaku, joka on erotettu omalle välilehdelleen hakuikkunassa. Päivämäärähaku kohdistuu vain tietokantaan, ja sillä voi hakea prosesseja ja objekteja. Prosesseja haettaessa voidaan syöttää päivämäärä ja valita, haetaanko ennen kyseistä päivää vai sen jälkeen tapahtuneet prosessit. Tietyllä aikavälillä tapahtuneet prosessit voidaan hakea syöttämällä myös toinen päivämäärä. Objekteja haettaessa käytettävissä ovat samat valinnat, mutta haku kohdistuu objektin aikaleimaan. Kuvassa 16 on kuvankaappaus päänäkymän yläosasta tilanteessa, jossa on valittu numerohaku.



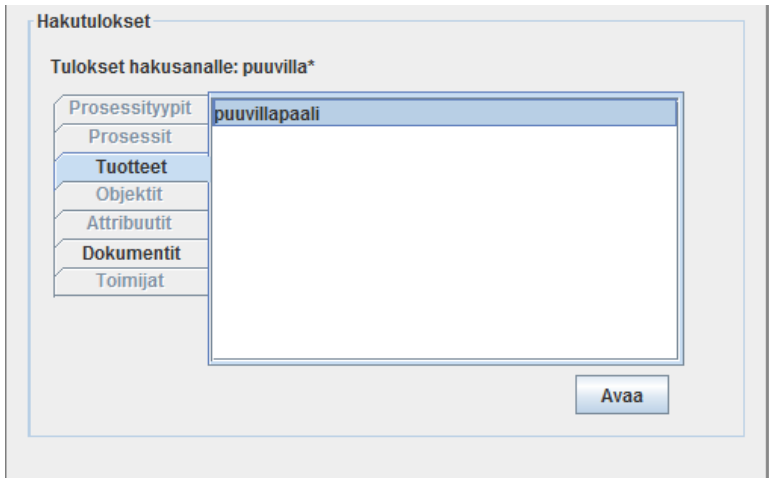
Kuva 16. Numerohaku.

Käyttäjän kirjoittama kysely välitetään hakunäkymäluokasta merkkijonona SearchHandler-luokalle. Jos merkkijono ei ole tyhjä sen jälkeen, kun sen alusta ja lopusta on poistettu tyhjät merkit, SearchHandler jatkaa kyselyn käsittelyä. Se kutsuu eri mallien hakumetodeja sen mukaan, millaisia olioita näkymässä on valittu haettavaksi. Malleille välitetään käsitelty merkkijono (hakuavain) ja haettavat ominaisuudet. Mallit välittävät kyselyn edelleen tietokantaan ja/tai käänteishakemistoon. Jos kysely tuottaa tuloksia, mallit ilmoittavat siitä tarkkailijoille, jotka päivittävät näkymän vastaamaan hakutuloksia.

Seuraavissa alakohdissa esitellään ohjelman ja tietokannan ja käänteishakemiston vuorovaikutusta esimerkkikyselyiden avulla. Kyselyn suorituksen kaikkia vaiheita ei kuvata, koska hausta riippumatta hakuavaimen välitys malleille tapahtuu aina edellisessä kappaleessa kuvatulla tavalla. Esimerkeissä keskitytäänkin lähinnä mallien toiminnan kuvaamiseen.

7.4.1. Esimerkkikysely: avainsanahaku

Kun halutaan tietää, miten jokin avainsana liittyy tuotteeseen tai prosesseihin, käytetään avainsanahakua. Avainsana voi esiintyä esimerkiksi tuotteen nimessä tai dokumentissa, joka on liitetty tuotteeseen tai prosessiin. Kun haetaan esimerkiksi hakuavaimella ”puuvilla*” kaikkia oliotyyppejä ja kaikilla ominaisuuksilla, saadaan kuvan 17 mukainen hakutulos. Haulla löytyy siis esimerkkitietokannasta tuotteita ja dokumentteja.



Kuva 17. Tulokset avainsanahaulle ”puuvilla*”.

Koska haku koskee kaikkia oliotyyppejä, ohjain kutsuu jokaisen mallin searchByKeyword()-metodia. Koodiesimerkissä 4 esitellään tuotteita käsittelevän ProductModel-luokan searchByKeyword()-metodin avulla, miten sanahaku tietokannasta toimii. Metodi saa parametreina merkkijonotyyppisen hakuavaimen ja luettelon ominaisuuksia, joihin haku kohdistuu.

```
    private final IndexManager indexManager = graphDb.index();
    private final Index<Node> productIndex = indexManager.forNodes("products");

    // Haku tietokannasta hakusanan perusteella.
    public void searchByKeyword(String searchKey, ArrayList<String> propertyKeys) {
        ArrayList<ProductImpl> results = new ArrayList<>();

        for (String propertyKey : propertyKeys) {

            IndexHits<Node> hits = productIndex.query(propertyKey, searchKey);

            //Luodaan kääreluokan oliot ja lisätään ne tuloslistaan.
            for (Node hit : hits) {
                 ProductImpl prod = new ProductImpl(hit, mainModel);

                 if (!results.contains(prod)) {
                        results.add(prod);
                    }
            }
        }
        //Ilmoitetaan tilan muutoksesta tarkkailijoille.
        this.setChanged();
        notifyObservers(results);
    }
```

Koodiesimerkki 4. Osa sanahaun suorittavasta metodista.

Esimerkkikoodin alussa otetaan käyttöön tietokannan tuotesolmuista luotu indeksi (productIndex-olio). Itse haku tehdään kutsumalla indeksiolion query()-metodia, jolle annetaan parametriksi merkkijonomuotoinen hakuavain ja haettavan ominaisuuden nimi. Metodi palauttaa iteraattorin, joka sisältää hakua vastaavat solmut. Solmuja kuvaavista Node-tyyppisistä olioista luodaan tuotetta kuvaavan kääreluokan oliot, jotka lisätään tulosluetteloon. Haku tehdään erikseen jokaiselle ominaisuudelle. Lopuksi metodi ilmoittaa mallin tarkkailijoille tilan muutoksesta, eli tässä tapauksessa haun suorittamisesta. Ilmoituksen yhteydessä tarkkailijoille välitetään lista hakutuloksista (hakutuloslista voi olla myös tyhjä). Tässä tapauksessa tarkkailija on päänäkymä, joka päivittyy vastaamaan hakutulosta. Koska haulla löytyi tuotteita, löytyneet tuotteet luetellaan omalla välilehdellään (kuva 17). Jos hakutuloslista olisi tyhjä, eli yhtään kyselyä vastaavaa tuotetta ei löytyisi, myös tuotevälilehti poistettaisiin käytöstä ja näytettäisiin harmaana.

Koodiesimerkissä 5 esitellään käänteishakemistoon tehtävistä kyselyistä vastaavan DocumentModel-luokan searchByKeyword()-metodin avulla, miten dokumentteja haetaan käänteishakemistosta.

```
// Haku käänteishakekmistosta hakusanan perusteella.
public void searchByKeyword(String searchKey, ArrayList<String> propertyKeys) {

      ArrayList<DocumentImpl> searchResultList = new ArrayList<>();
      Document doc;

       try {
           //Jäsennetään kysely.
           Query query = parser.parse(searchKey);

           //Haku.
           TopDocs results = searcher.search(query, 100);
       }
       catch (ParseException | IOException ex) { … }
       ScoreDoc[]hits = results.scoreDocs;

       // Käydään tulokset läpi.
       for (int i = 0; i < hits.length; i++) {
           try {
               doc = searcher.doc(hits[i].doc);
           }
           catch (IOException ex) { … }

           // Lisätään tulosdokumentti tarkkailijoille lähetettävään listaan.
           searchResultList.add(new DocumentImpl(doc, mainModel));
       }
       // Ilmoitetaan tilan muutoksesta tarkkailijoille.
       setChanged();
       notifyObservers(searchResultList);
   }
```

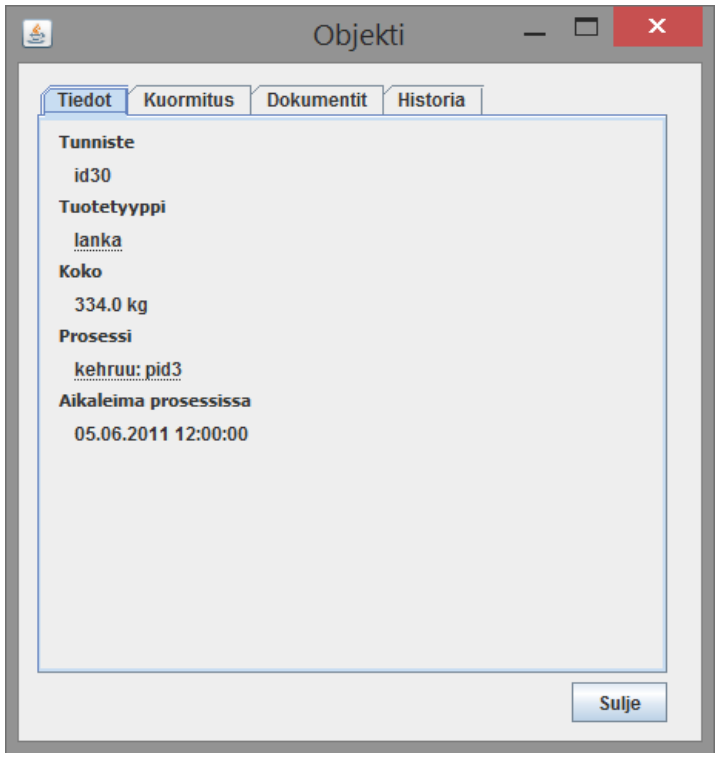
Koodiesimerkki 5. Avainsanahaku käänteishakemistosta.

Metodille välitettävästä merkkijonomuotoisesta hakuavaimesta luodaan Lucenen omalla jäsentimellä Query-olio, joka välitetään kyselyn suorituksesta vastaavan IndexSearcher-tyyppisen olion (koodin searcher-olio) hakumetodille. Haun tuloksena on joukko dokumentteja, jotka täsmäävät kyselyyn. Koska hakutulokset palautetaan Lucenen Document-olioina, niistä luodaan

DocumentImpl-tyyppiset kääreoliot. Lopuksi ilmoitetaan tarkkailijoille, että uusi hakutuloslista on käytettävissä.

7.4.2. Esimerkkikysely: objektin tiedot

Kuvassa 18 on ikkuna, joka avautuu, kun hakutuloksista valitaan objekti, jonka tiedot halutaan näyttää. Tässä esimerkissä tarkasteltavaksi on valittu lankarulla, jonka tunniste on id30.



Kuva 18. Objektin tiedot.

Objekti-ikkunan ensimmäisellä välilehdellä näytetään objektin perustiedot. Toisella välilehdellä luetellaan objektiin liittyvät tulos- ja syöteattribuutit ja kolmannella välilehdellä siihen liittyvät objektit. Historia-välilehti sisältää visualisoinnin objektin alkuperästä, eli objekteista, joista objekti koostuu.

Kuvassa 18 objektin tuotetyyppi ja alkuperäprosessi on alleviivattu. Alleviivaus tarkoittaa, että tuotetyyppi ja prosessi ovat linkkejä, joita napsauttamalla voi siirtyä tarkastelemaan alleviivatun olion tietoja. Esimerkiksi ”kehruu: pid3” -linkin napsauttaminen avaisi ikkunan, joka sisältää prosessin pid3 tiedot ja siihen liittyvät objektit, kuormitustiedot ja dokumentit. Vastaavia linkkejä on myös muiden olioiden tiedot esittävissä ikkunoissa. Näin olioihin liittyviä tietoja voidaan selata näkymien avulla ilman, että esimerkiksi objektiin liittyvän prosessin attribuuteista tarvitsisi tehdä uutta hakua.

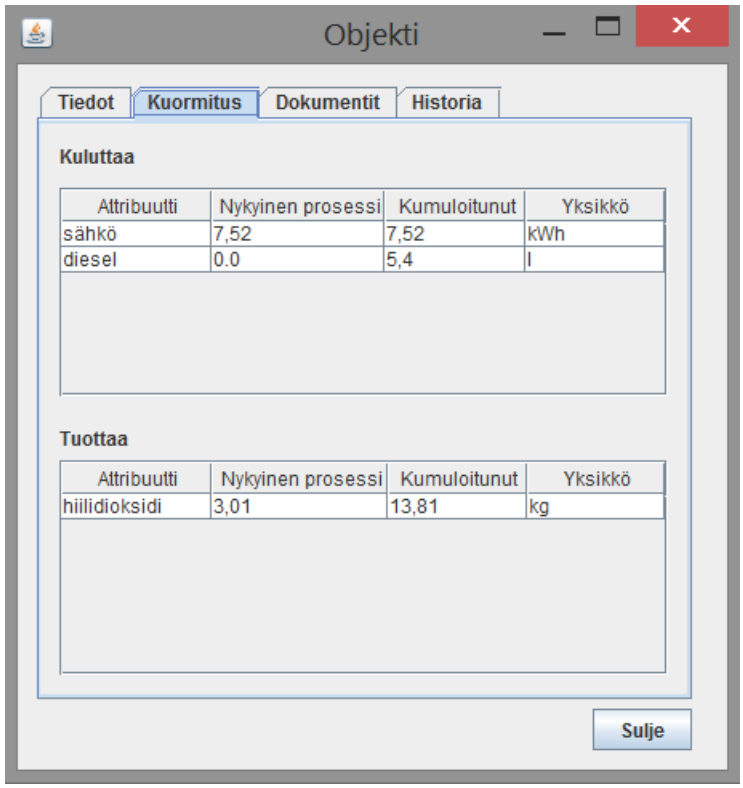
Koodiesimerkissä 6 on otteita kuvan 18 näkymän koostamiseen tarvittavasta koodista. Esimerkissä haetaan ensin objektia edustavan solmun nimi, eli ”id30”. Objektiin liittyvä tuote voidaan selvittää etsimällä objektista lähtevä IS_A-suhde ja hakemalla suhteen loppusolmu, eli tuotesolmu, jonka nimi on ”lanka”.

```
Node object;

// Haetaan solmun ”name”-ominaisuuden arvo.
String name = (String) node.getProperty(”name”);

// Haetaan tuote, jonka esiintymä objekti on.
Relationship rs = object.getSingleRelationship(RelationshipTypes.IS_A,
          Direction.OUTGOING);
Node product = rs.getOtherNode(object);
```

Koodiesimerkki 6. Solmun ominaisuuden hakeminen ja solmusta lähtevän suhteen ja sen loppusolmun selvittäminen.

7.4.3. Esimerkkikysely: attribuuttien arvon laskeminen

Kuvassa 19 on edellisessä alikohdassa kuvatun ikkunan toinen välilehti, jolla näkyvät objektin (id30) kuormitustiedot. Syöte- ja tulosattribuutit luetellaan omissa taulukoissaan, ja molemmista ilmoitetaan sekä objektin valmistusprosessin arvo että attribuutin kumuloitunut arvo. Valmistusprosessi, eli nykyinen prosessi, tarkoittaa olion tuottanutta prosessia.



Kuva 19. Objektiin liittyvät syöte- ja tulosattribuutit.

Kun lasketaan objektin osuutta sen valmistusprosessin attribuuteista, lasketaan ensin objektin osuus prosessin kuormituksesta jakamalla objektin koko, tässä tapauksessa paino, kaikkien prosessiin osallistuvien samaa tuotetta olevien objektien yhteenlasketulla koolla (esimerkiksi 334 kg / 1 000 kg = 0,334). Sen jälkeen haetaan attribuutit, joita objektin valmistusprosessiin liittyy. Esimerkiksi kehruuprosessissa pid3, jossa tämän esimerkin lankarulla on valmistettu, kuluu 25

kWh sähköä. Tämä määrä kerrotaan objektin koosta lasketulla suhdeluvulla ja objektiin liittyvän tuotteen osuudella prosessin kustannuksista (langan osuus kehruuprosessin kustannuksista on 90 %). Näin tähän objektiin liittyväksi sähkönkulutukseksi saadaan $25 \cdot 0{,}334 \cdot 0{,}9 = 7{,}52$ kWh.

Objektin prosessikohtaista kuormitusta tarvitaan, kun lasketaan objektin kumuloituneita attribuuttien arvoja. Kumuloituneilla arvoilla tarkoitetaan kuormitusta, joka objektiin kohdistuu koko sen elinkaaren ajalla. Niiden laskemisessa otetaan huomioon kaikki objektit, joista objekti koostuu. Koodiesimerkissä 7 kuvataan, miten objektille lasketaan kumuloitunut attribuuttikuorma.

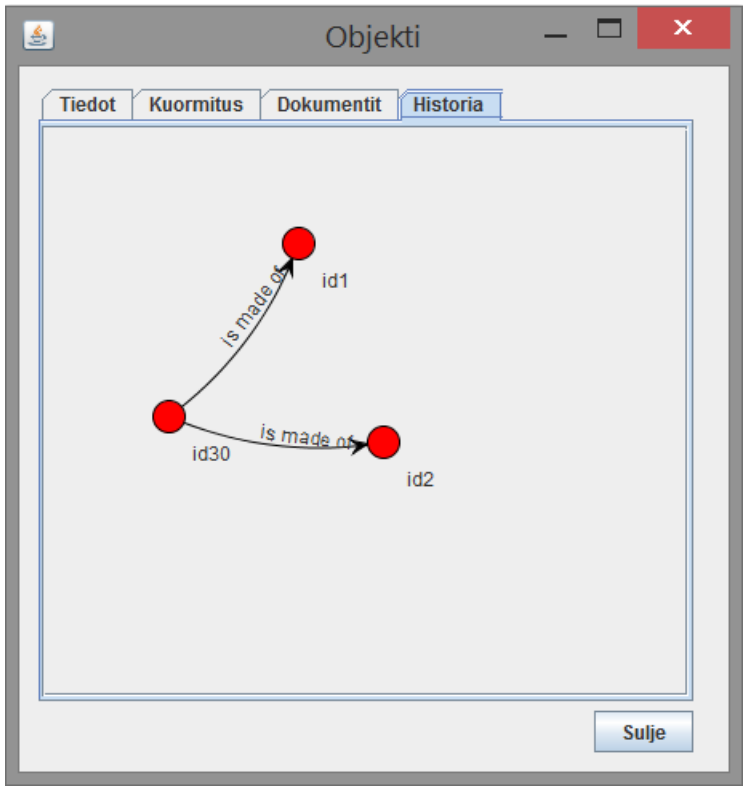
```
public Map<AttributeImpl, Double> getCumulatedBurden(Node object,
          RelationshipTypes rsType) {

   Map<AttributeImpl, Double> burden = this.getBurden(object, rsType);

   Iterable<Relationship> parentRelationships =
          object.getRelationships(RelationshipTypes.IS_MADE_OF,
          Direction.OUTGOING);

   // Haetaan objektin vanhempien kumuloituneet arvot.
   for (Relationship parentRs : parentRelationships) {

     Double transFunction = (Double)
          parentRs.getProperty(”transformationFunction”);
     Node parent = parentRs.getOtherNode(object);
     Iterator<AttributeImpl> bKeys = burden.keySet().iterator();

     Map<AttributeImpl, Double> parentBurden = getCumulatedBurden(parent,
                                                                  rsType);

      // Lasketaan uusi arvo kaikille kumuloituville attribuuteille.
      for(AttributeImpl key : parentBurden.keySet()) {

         Double parValue = parentBurden.get(key);
         Double ownValue;
         boolean found = false;

         // Etsitään vastaava attribuutti objektin valmistusprosessin
         // attribuuteista.
         while (bKeys.hasNext() && found == false) {
             AttributeImpl next = bKeys.next();

             if (next.getName().equalsIgnoreCase(key.getName())) {
                ownValue = burden.get(next);
                // Lasketaan attribuuttien arvot yhteen.
                burden.put(next, ownValue+(parValue*transFunction));
                found = true;
             }
         }
         // Jos attribuuttia ei kulu nykyisessä prosessissa, lisätään uusi arvo.
         if (found == false) {
            burden.put(key, parValue*transFunction);
         }
      }
  }
  return burden;
}
```

Koodiesimerkki 7. Objektin kumuloituneen attribuuttikuorman laskeminen.

Koodiesimerkin metodi saa parametrina objektin, jonka kuormitus haetaan, ja suhdetyypin (USES, kun ollaan kiinnostuneita syöteattribuuteista, ja PRODUCES, kun ollaan kiinnostuneita tulosattribuuteista). Koodiesimerkissa etsitään aluksi kaikki objektista lähtevät IS_MADE_OF-suhteet. Niiden loppusolmut ovat objektin välittömät vanhemmat, eli ne objektit, joista objekti koostuu. Jokaiselle objektin vanhemmalle lasketaan kumuloitunut attribuuttikuorma kutsumalla getCumulatedBurden()-metodia rekursiivisesti. Näin laskentaan saadaan mukaan objektin koko alkuperäketju. Objektien vanhempien kumuloitunut attribuuttikuorma lasketaan yhteen objektin nykyisen attribuuttikuorman kanssa. Yhteenlaskussa käytetään ”transformationFunction”-ominaisuutta, jonka arvon laskentatapa on selostettu kohdassa 5.3. Jos objektien vanhempien attribuuttikuorma sisältää attribuutteja, joita objektin valmistusprosessissa ei kulu, lisätään niiden arvo objektin kuormaan pelkästään ”transformationFunction”-arvolla kerrottuna.

7.4.4. Esimerkkikysely: objektin alkuperä

Objektin alkuperällä viitataan tässä objekteihin, joita on käytetty objektin valmistamiseen. Kuvassa 20 näkyy edellisissä alikohdissa käsitellyn lankarullan (id30) alkuperäketju. Alkuperäketju esitetään visuaalisesti, jotta se olisi helpompi hahmottaa myös niissä tapauksissa, joissa objektilla on enemmän vanhempia. Visualisoinnin luomiseen on käytetty JUNG-ohjelmistokirjastoa, joka on tarkoitettu graafi- tai verkkomuotoisen tiedon mallintamiseen, analysointiin ja visualisointiin [JUNG, 2015].



Kuva 20. Objektin alkuperäketju.

Objektin alkuperä selvitetään ohjelmassa luomalla TraversalDescription-tyyppinen kuvaus, jossa määritetään, millaista reittiä tietokannasta (eli graafista) etsitään. Kuvaus määritetään aina lähtösolmusta katsoen. Koodiesimerkissä 8 on kuvaus, jota on käytetty kuvan 20 visualisoinnin luomisessa. Kuvauksessa määritetään ensin, että solmusta lähdetään kulkemaan solmusta lähteviä IS_MADE_OF-tyyppisiä suhteita pitkin. Seuraavaksi määritetään, että suhteita kuljetaan vain syvyyteen ”1”, eli ei kuljeta yhtä suhdetta pidemmälle. Lisäksi kerrotaan, että lähtösolmua ei oteta mukaan tuloksiin (objekti ei ole itsensä vanhempi).

Kulkeminen tapahtuu TraversalDescription-olion traverse()-metodia kutsumalla. Se saa parametrina kulkemisen lähtösolmun ja palauttaa joukon solmuja, jotka löytyvät, kun tietokannassa kuljetaan kuvauksessa määritetty reitti. Esimerkiksi solmun id30 tapauksessa palautetaan solmut id1 ja id2, jotka kuvaavat lankarullaobjektin id30 valmistuksessa käytettäviä puuvillapaaliobjekteja.

```
Node object;
ResourceIterable<Node> parents;

parents = graphDb.traversalDescription()
.relationships(RelationshipTypes.IS_MADE_OF, Direction.OUTGOING)
.evaluator(Evaluators.toDepth(1))
.evaluator(Evaluators.excludeStartPosition())
.traverse(object).nodes();
```

Koodiesimerkki 8. Objektin lähimpien vanhempien haku.

Solmun kaikki vanhemmat voidaan hakea jättämällä koodiesimerkin 8 kuvauksesta pois rivi, jossa rajataan kulkemisen syvyys ensimmäiseen solmusta lähtevään suhteeseen. Tällöin lähtösolmusta lähteviä IS_MADE_OF-suhteita seurataan niin pitkälle kuin mahdollista.

7.4.5. Esimerkkikysely: päivämäärähaku

Numerohaussa objekteja ja prosesseja voidaan hakea päivämäärän perusteella. Se mahdollistaa esimerkiksi jollain aikavälillä valmistettujen objektien hakemisen. Päivämäärät syötetään käyttöliittymässä muodossa pp/kk/vvvv, mutta ne on tallennettu tietokantaan numeerisessa muodossa. Siksi SearchHandler-luokan olio muuntaa käyttäjän syöttämät päivämäärät tietokannassa käytettyyn Long-muotoon, ennen kuin se välittää ne mallin hakumetodille. Koodiesimerkissä 9 merkkijonomuodossa oleva päivämäärä muunnetaan Long-tyyppiseksi olioksi.

```
String startDate = ”01/06/2013”;
Long startTime =
 new java.text.SimpleDateFormat("dd/MM/yyyy").parse(startDate).getTime()/1000;
```

Koodiesimerkki 9. Päivämäärän muuntaminen tietokannassa käytettyyn muotoon.

Päivämäärähaku kohdistuu suhteista muodostettuun indeksiin, koska prosessiin osallistuvien objektien aikaleima on tallennettu prosessin ja objektin välisen ROUTE-suhteen ominaisuudeksi. Koodiesimerkissä 10 esitetään, miten objekteja haetaan tietokannasta aikaleiman perusteella.

```
private final Index<Relationship> routeIndex =
    indexManager.forRelationships("relRouteTime-fulltext");

public Map<Node, Long> getObjectsByInterval(Long start, Long end) {

   Map<Node, Long> objects = new HashMap<>();

   IndexHits<Relationship> hits =
    routeIndex.query(QueryContext.numericRange("time", start, end));

   for (Relationship hit : hits) {
        objects.put(hit.getStartNode(), (Long) hit.getProperty("time"));
   }
   return objects;
}
```

Koodiesimerkki 10. Objektien haku päivämäärän perusteella.

Numerohaku toimii samalla tavalla kuin aiemmin esitetty sanahaku. Haluttuun indeksiin tehdään kysely, joka sisältää halutun ominaisuuden avaimen ja arvon, jota etsitään. Koodiesimerkissä 10 hyödynnetään Lucenen mahdollistamaa hakutyyppiä, jossa haetaan kohteet, joiden ”time”-ominaisuuden arvo on annetulla välillä. Kyselymetodin parametrit ovat ominaisuus, johon kysely kohdistuu, sekä haettavan numerovälin suurin ja pienin arvo. Haku kohdistetaan ROUTE-tyyppisistä suhteista muodostettuun indeksiin.

Koska kyselyn tuloksena on joukko suhteita, mutta haku kohdistui varsinaisesti objekteihin, täytyy tulosjoukon pohjalta muodostaa uusi joukko, joka sisältää suhteisiin liittyvät objektit. Koska tulosjoukon suhteet ovat ROUTE-tyyppisiä suhteita, jotka suuntautuvat tietokannassa aina objektista prosessiin, voidaan halutut objektit hakea etsimällä suhteen alkusolmu.

8. Yhteenveto

Monilla sovellusalueilla, kuten paikannuksessa, sosiaalisissa verkostoissa ja webissä, tieto on verkottunutta, eli se sisältää paljon tietoa kuvaavien olioiden keskinäisiä suhteita, joiden analysointi on yhtä tärkeää kuin itse olioiden ja olioihin liittyvien arvojen. Perinteiset tietomallit on kuitenkin suunniteltu arvojen käsittelyn näkökulmasta, ja olioiden keskinäisten suhteiden käsittely on niillä monimutkaista.

Graafitietomalli on jo 1980-luvulla alkunsa saanut tietomalli, joka on viime vuosina noussut uudelleen tutkimuksen kohteeksi, koska se ottaa huomioon nimenomaan tiedon verkkomaisen rakenteen ja mahdollistaa suhteiden sujuvamman käsittelyn. Tässä työssä esiteltiin Neo4j-tietokanta, joka on tällä hetkellä suosituin graafitietomalliin pohjautuvista tietokannoista.

Neo4j-tietokannan ominaisuuksia havainnollistettiin toimitusketjujen mallintamiseen tarkoitetun jäljitettävyysgraafin avulla. Koska jäljitettävyysgraafi on graafi, sen tallentaminen graafitietokantaan on luontevaa. Prosessien lisäksi sekä tuotteet että objektit voidaan kuvata tietokantagraafin solmuina, ja prosesseissa käsiteltävien objektien keskinäiset yhteydet voidaan esittää solmujen välisinä suhteina. Kun objektien muuntuminen toisiksi objekteiksi tallennetaan tietokantaan suhteiden avulla, voidaan esimerkiksi objektin alkuperä selvittää helposti graafissa kulkemalla.

Jäljitettävyysgraafi toteutettiin graafitietokannassa, ja tietokannan ympärille ohjelmoitiin Javalla hakuohjelma, jonka kautta tietokantakyselyt tehdään. Lisäksi jäljitettävyysgraafiin liittyvistä dokumenteista muodostettiin käänteishakemisto, joka yhdistettiin graafitietokantaan. Näin tekstitiedonhaku voitiin yhdistää tietokantaan tallennettujen jäljitettävyystietojen hakuun. Tiedonhakua havainnollistettiin muutaman esimerkkikyselyn avulla. Niillä osoitettiin, miten graafitietokannasta voidaan hakea prosesseihin ja tuotteisiin liittyviä tietoja.

9. Johtopäätökset

Tässä työssä graafitietokantaa käsiteltiin pelkästään Java-ohjelmointirajapinnan operaatioilla. Sen ansiosta tietokantagraafia ja sen solmuja ja suhteita voitiin käsitellä ohjelmassa Java-olioina, joten tietokanta- ja ohjelmakoodin yhdistäminen oli helppoa. Näin olioiden käsittely oli yksinkertaista, kun esimerkiksi tietokantasolmujen ominaisuuksia voitiin hakea olioiden metodien kautta. Samoin graafissa kulkemisessa käytettävien kuvausten laatiminen oli helppoa ohjelmointirajapinnan operaatioiden avulla, kun kuvaus voitiin määritellä luettelemalla erilaisia rajoituksia ja ohjeita, joiden mukaan graafissa kuljetaan. Myös sanahaku käänteishakemistosta voitiin yhdistää suoraan muuhun koodiin, koska Lucene-hakukirjasto on toteutettu Javalla.

Tietokantaolioiden käsittely vaati esimerkiksi attribuuttien arvojen laskennassa välillä monivaiheisiakin operaatioita, kun ensin jouduttiin hakemaan tietyntyyppinen objektiin liittyvä suhde ja sen jälkeen toinen suhteeseen liittyvä solmu. Toinen vaihtoehto olisi ollut tehdä tietokantakyselyt Cypher-kyselykielellä, ja olisikin kiinnostavaa, millainen kysely esimerkiksi edellä mainitusta attribuuttien arvojen laskennasta muodostuisi. Oletettavasti kyselykieltä hyödyntävä lähestymistapa toimisi olioiden käsittelyä paremmin sovelluksissa, joissa kyselyiden pääpaino on tietokannan tietojen analysoinnissa, kuten erilaisten prosessikohtaisten lukujen laskemisessa tai olioiden hakemisessa tiettyjen ominaisuuksien perusteella. Tässä työssä painotus oli kuitenkin olioiden välisten suhteiden tutkimisessa, ja siihen tarkoitukseen oliopohjainen lähestymistapa sopi hyvin.

Kyselymenetelmästä riippumatta jäljitettävyysgraafi vaikuttaisi sopivan hyvin graafitietokantaan tallennettavaksi. Toimitusketjuun liittyvät prosessit ja tuotteet ja niiden väliset suhteet voidaan kuvata tietokannassa graafirakenteena, joka vastaa hyvin sitä, millaisia verkostoja toimitusketjut tosimaailmassakin muodostavat. Näin toimitusketjun verkkomainen rakenne voidaan siirtää tietokantaan sellaisenaan ilman, että sitä tarvitsee ensin muuntaa jonkin muun tietomallin käsitteistön mukaiseksi.

Graafin visualisointia käsiteltiin tässä työssä lähinnä esimerkinomaisesti objektin alkuperäketjun yhteydessä, vaikka yksi graafitietomallin vahvuuksista on juuri siinä, että graafeja voidaan havainnollistaa helposti visuaalisesti. Toinen kiinnostava jatkokehittelyn kohde olisikin graafisen käyttöliittymän hyödyntäminen kattavammin niin, että esimerkiksi objektin tietoja näytettäessä objektiin liittyvät muut oliot ja niiden suhteet tuotaisiin esiin visualisoinnin avulla.

Viiteluettelo

[Andries *et al.*, 1992] Marc Andries, Marc Gemis, Jan Paxedaens, Inge Thyssens and Jan Van den Bussche, Concepts for graph-oriented object manipulation. In: *Advances in Database Technology – EDBT '92, Lecture Notes in Computer Science* **580** (1992), Springer, 21–38.

[Angles and Gutierrez, 2008] Renzo Angles and Claudio Gutierrez, Survey of graph database models. *ACM Comput. Surv*. **40**, 1 (February 2008), 2–39.

[Apache Lucene API, 2015a] Apache Lucene API: Document. Saatavilla: http://lucene.apache.org/core/5_0_0/core/org/apache/lucene/document/Document.html Viitattu 23.3.2015.

[Apache Lucene API, 2015b] Apache Lucene API: Query Parser Syntax. Saatavilla: http://lucene.apache.org/core/5_0_0/queryparser/org/apache/lucene/queryparser/classic/package-summary.html#package_description Viitattu 7.4.2015.

[Apache Lucene Core, 2015] The Apache Software Foundation, Apache Lucene Core. Saatavilla: https://lucene.apache.org/core/ Viitattu 23.3. 2015.

[Aurelius, 2015] Aurelius: Titan. Saatavilla: http://thinkaurelius.github.io/titan/ Viitattu 30.4.2015.

[Batra and Tyagi, 2012] Shalini Batra and Charu Tyagi, Comparative analysis of relational and graph databases. *IJSCE* **2**, 2 (May 2012), 509–512.

[Bechini *et al.*, 2005] Alessio Bechini, Mario Cimino, Beatrice Lazzerini, Francesco Marcelloni and Andrea Tomasi, A general framework for food traceability. In: *Proc. of the 2005 Symposium on Applications and the Internet Workshops (SAINT-W'05)*, 366–369.

[Bechini *et al.*, 2008] Alessio Bechini, Mario Cimino, Francesco Marcelloni and Andrea Tomasi, Patterns and technologies for enabling supply chain traceability through collaborative e-business. *Inform. Software. Tech.* **50** (2008), 342–359.

[Blackhurst et al, 2005] Jennifer Blackhurst, Tong Wu and Peter O'Grady, PCDM: a decision support modeling methodology for supply chain, product and process design decisions. *J. Oper. Manag.* **23** (2005), 325–343.

[Buerli, 2012] Mike Buerli, The current state of graph databases. Saatavilla: http://bigbe.su/lectures/2014/16.3.pdf Viitattu 23.4.2015.

[Chen, 1976] Peter Pin-Shan Chen, The Entity-relationship model: toward a unified view of data. *TODS* **1**, 1 (March 1976), 9–36.

[Codd, 1970] Edgar F. Codd, A relational model of data for large shared data banks. *Comm. ACM* **13**, 6 (June 1970), 377–387.

[Das *et al.*, 2014] Souripriya Das, Jagannathan Srinivasan, Matthew Perry, Eugene Chong and Jayanta Banerjee, A tale of two graphs: property graphs as RDF in Oracle. In: *Proc. of 17th International Conference on Extending Database Technology*, *EDBT 2014*. 762–773.

[DB-Engines.com, 2015] DB-Engines.com, DB-Engines Ranking of Graph DBMS. Saatavilla: db-engines.com/en/ranking/graph+dbms Viitattu 11.5.2015.

[DBTG, 1971] Data base task group report to the CODASYL programming language committee, ACM, 1971.

[Dominguez-Sal *et al.*, 2010] David Dominguez-Sal, Pere Urbón-Bayes, Aleix Giménez-Vañó, Sergio Gómez-Villamor, Norbert Martínez-Bazan and Josep-Lluis Larriba-Pey, Survey of graph database performance on the HPC scalable graph analysis benchmark. In: *Proc. of the 2010 international conference on Web-age information management*, 37–48.

[Elmasri and Navathe, 1994] Ramez Elmasri and Shamkant B. Navathe, *Fundamentals of Database Systems (Second Edition)*. Benjamin Cummings, 1994.

[Elmasri and Navathe, 2010] Ramez Elmasri and Shamkant B. Navathe, *Fundamentals of Database Systems (Sixth Edition)*. Benjamin Cummings, 2010.

[EUR-Lex, 2013a] Euroopan parlamentin ja neuvoston direktiivi 2001/95/EY, annettu 3 päivänä joulukuuta 2001, yleisestä tuoteturvallisuudesta. Saatavilla: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32001L0095:FI:NOT Viitattu 16.12.3013.

[EUR-Lex, 2013b] Euroopan parlamentin ja neuvoston asetus (EY) N:o 178/2002, annettu 28 päivänä tammikuuta 2002, elintarvikelainsäädäntöä koskevista yleisistä periaatteista ja vaatimuksista,. Saatavilla: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32002R0178:FI:NOT Viitattu 16.12.2013.

[Evira, 2011] Evira, Elintarvikkeiden jäljitettävyys. Saatavilla: http://www.evira.fi/portal/fi/elintarvikkeet/valmistus+ja+myynti/jaljitettavyys+/ Viitattu 3.2.2015.

[Grolinger *et al.*, 2013] Katarina Grolinger, Wilson Higashino, Abhinav Tiwari and Miriam Capretz, Data management in cloud environments: NoSQL and NewSQL data stores. *JoCCASA* **2**:22 (December 2013).

[Gyssens *et al.*, 1990] Marc Gyssens, Jan Paredaens and Dirk Van Gucht, A graph-oriented object database model. In: *Proceedings of the 9th symposium on Principles of database systems*. ACM, 417–424.

[Hammer and McLeod, 1981] Michael Hammer and Dennis McLeod, Database description with SDM: a semantinc database model. *TODS* **6**, 3 (September 1981), 351–386.

[Handfield and Nichols, 1999] Robert Handfield and Ernest Nichols, *Introduction to Supply Chain Management*, Prentice Hall, 1999.

[Helsingin Sanomat, 2014] Helsingin Sanomat 20.5.2014, Autojätti GM kutsuu jälleen miljoonia autoja korjattavaksi vikavyyhdissä. Saatavilla: http://www.hs.fi/talous/a1400568019059 Viitattu 10.5.2015.

[Jansen-Vullers *et al.*, 2003] Monique Jansen-Vullers, Christian van Dorp, Adriaan Beulens, Managing traceability information in manufacture. *Int. J. Inform. Manage*. **23** (2003), 395–413.

[JUNG, 2015] JUNG – the Java Universal Network/Graph Framework. Saatavilla: http://jung.sourceforge.net/ Viitattu 10.5.2015.

[Junkkari and Sirkka, 2011a] Marko Junkkari and Antti Sirkka, Using RFID for tracing cumulated resources and emissions in supply chain, *Int. J. Ad Hoc and Ubiquitous Computing* **8**, 4 (2011), 220–229.

[Junkkari and Sirkka, 2011b] Marko Junkkari and Antti Sirkka, Formal definition of traceability graph. University of Tampere, School of Information Sciences, Reports in Information Sciences 3, November 2011.

[Junkkari and Sirkka, 2014] Marko Junkkari and Antti Sirkka, Data-centric workflowa to lifecycle data management. In: *Proceedings of 3rd International Conference on Data Management Technologies and Applications (DATA2014)*, 116–124.

[Kim *et al.*, 1995] Henry Kim, Mark Fox and Michael Gruninger, An ontology of quality for enterprise modelling. In: *Proc. of the Fourth Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET-ICE)*, 1995, 105–116.

[Kuper and Vardi, 1984] Gabriel Kuper and Moshe Vardi, A new approach to database logic. In: *Proceedings of the 3th Symposium on Principles of Database Systems (PODS)*. ACM Press, 86–96.

[Lécluse *et al.*, 1988] Christophe Lécluse, Philippe Richard and Fernando Velez, O_2, an object-oriented data model. In: *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*. ACM, 424–433.

[Levene and Loizou, 1995] Mark Levene and George Loizou, A graph-based data model and its ramifications. *IEEE Trans. Knowl. Data Eng.* **7**, 5 (October 1995), 809–823.

[Moe, 1998] Tina Moe, Perspectives on traceability in food manufacture. *Trends Food Sci. Tech.* **9** (1998), 211–214.

[Neo4j.com, 2015] Neo4j: What is Neo4j? Saatavilla: http://neo4j.com/developer/graph-database/#_what_is_neo4j Viitattu 15.3.2015.

[Neo4j API 2015a] Neo4j API: Relationship. Saatavilla: http://neo4j.com/docs/2.2.0/javadocs/org/neo4j/graphdb/Relationship.html Viitattu 13.3.2015.

[Neo4j API 2015b] Neo4j API: Relationship type. Saatavilla: http://neo4j.com/docs/2.2.0/javadocs/org/neo4j/graphdb/RelationshipType.html Viitattu 13.3.2015.

[Neo4j API 2015c] Neo4j API: PropertyContainer. Saatavilla: http://neo4j.com/docs/2.2.0/javadocs/org/neo4j/graphdb/PropertyContainer.html Viitattu 13.3.2015.

[Neo4j API 2015d] Neo4j API: Node. Saatavilla: http://neo4j.com/docs/2.2.0/javadocs/org/neo4j/graphdb/Node.html Viitattu 8.5.2015.

[Neo4j Manual, 2015a] The Neo4j Manual: The Traversal Framework, Main Concepts. Saatavilla: http://neo4j.com/docs/stable/tutorial-traversal-concepts.html Viitattu 13.3.2015.

[Neo4j Manual, 2015b] The Neo4j Manual: Schema. Saatavilla: http://neo4j.com/docs/stable/graphdb-neo4j-schema.html Viitattu 13.3.2015.

[Neo4j Manual, 2015c] The Neo4j Manual: Cypher Query Language. Saatavilla: http://neo4j.com/docs/stable/cypher-query-lang.html Viitattu 10.4.2015.

[Neo4j Manual, 2015d] The Neo4j Manual: Extra features for Lucene indexes. Saatavilla: http://neo4j.com/docs/stable/indexing-lucene-extras.html Viitattu 10.5.2015.

[Objectivity, 2015] Objectivity: InfiniteGraph. Saatavilla: http://www.objectivity.com/infinitegraph Viitattu 30.4.2015.

[OODesign, 2015] Object-oriented design: Observer Pattern. Saatavilla: http://www.oodesign.com/observer-pattern.html Viitattu 8.5.2015.

[Opara, 2003] Linus Opara, Traceability in agriculture and food supply chain: a review of basic concepts, technological implications, and future prospects. *Journal of Food Agriculture and Environment* **1**, 1, 101–106.

[Robinson *et al.*, 2013] Ian Robinson, Jim Webber and Emil Eifrem, *Graph Databases*, O'Reilly Media, 2013.

[Roussopoulos and Mylopoulos, 1975] Nicholas Roussopoulos and John Mylopoulos, Using semantic networks for data base management. In: *Proceedings of the 1st International Conference on Very Large Data Bases*. ACM, 144–172.

[Saarinen, 2013] Juhani Saarinen, Tieto ruuan alkuperästä yhä tärkeämpi ostoperuste. Helsingin sanomat 20.6.2013. Saatavilla: http://www.hs.fi/talous/a1371694530200 Viitattu 16.12.2013

[Sahin and Robinson, 2002] Funda Sahin and E. Powell Robinson, Flow coordination and information sharing in supply chains: review implications and directions for future research. *Decision Sci.* **33**, 4, 505–536.

[Senneset *et al.*, 2007] Gunnar Senneset, Eskil Forås and Kari M. Fremme, Challenges regarding implementation of electronic chain traceability. *Brit. Food. J.* **109**, 10 (2007), 805–818.

[Shapiro, 2009] Jeremy Shapiro, *Modeling the Supply Chain*, Cengage Learning, 2009.

[Shipman, 1981] David Shipman, The functional data model and the data languages DAPLEX. *TODS* **6**, 1 (March 1981), 140–173.

[Sibley and Kerschberg, 1977] Edgar Sibley and Larry Kerschberg, Data architecture and data model considerations. In: *Proc. National Computer Conference, AFIPS* (June 1977), 85–96 .

[Silberschatz *et al.*, 1996] Avi Silberschatz, Henry F. Korth and S. Sudarshan, Data models. *CSUR* **28**, 1 (March 1996), 105–108.

[Stonebraker et al, 2007] Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem and Pat Helland, The end of an architectural era: (it's time for a complete rewrite). *In Proceedings of the 33rd international conference on Very large data bases*, ACM, 1150–1160.

[Taloussanomat, 2013] Taloussanomat 11.4.2013, Turvatyynyvikoja? – Toyota, Honda ja Nissan vetävät autoja takaisin. Saatavilla: http://www.taloussanomat.fi/autot/2013/04/11/turvatyyny vikoja-toyota-honda-ja-nissan-vetavat-autoja-takaisin/20135254/304 Viitattu 10.5.2015.

[Thompson *et al.*, 2005] Michael Thompson, Gilbert Sylvia and Michael Morrissey, Seafood traceability in the United States: current trends, system design, and potential applications. *Compr. Rev. Food Sci. F.* **1** (2005), 1–7.

[Tsichritzis and Lochovsky, 1976] Dennis C. Tsichritzis and Frederick Lochovsky, Hierarchical database management: a survey. *CSUR* **8**, 1 (March 1976), 105–123.

[Tukes, 2013] Valtioneuvoston asetus kulutustavaroista ja kuluttajapalveluksista annettavista tiedoista. Saatavilla: http://plus.edilex.fi/tukes/fi/lainsaadanto/20040613?toc=1 Viitattu 16.12.2013.

[Ullman, 1988] Jeffrey D. Ullman, *Principles of Database and Knowledge-Base Systems, Volume I: Classical Database Systems*. Computer Science Press, Rockville, 1988.

[Vicknair *et al.*, 2010] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen and Dawn Wilkins, A comparison of a graph database and a relational database. In: *ACMSE '10 Proc. of the 48th Annual Southeast Regional Conference*, Article No. 42.

[Yle, 2013] Anniina Wallenius, Hevosenlihaa löytyi lasagnesta myös Sveitsissä, Saksassakin epäilyksiä. Saatavilla: http://yle.fi/uutiset/hevosenlihaa_loytyi_lasagnesta_myos_sveitsissa_saksassakin_epailyksia/6494576 Viitattu 16.12.3013.

[Yu *et al.*, 2003] H. Yu, A. Reyes, S. Cang and S. Lloyd, Combined Petri net modeling and AI based heuristic hybrid search for flexible manufacturing systems – part 1. *Computers and Industrial Engineering* **44**, 4, 527–543.

[Zurawski and Zhou, 1994] Richard Zurawski and MengChu Zhou, Petri nets and industrial applications: a tutorial. *IEEE Trans. Ind. Electron.* **41**, 6 (December 1994), 567–583.